Check Program

This tutorial will explain 'Check Program' utility that checks the correctness of an Alphabets program through static analysis. The rules 'Check Program' uses are described in “Hierarchical Static Analysis of Structured Systems of Affine Recurrence Equations” by Dupont de Dinechin and Robert [ASAP 1996]

Usage

```
prog = ReadAlphabets("../../alphabets//matrix_product.ab");
CheckProgram(prog);
```

The main analyses performed by Check Program are described below. They exploit the algorithms for constructing domains and context domains of Alphabets expressions.

Completeness Analysis

First, Check program ensures that each local or output variable has an equation, and that the variable is is defined for all points in its declared domain. In other words, each equation must ***completely*** define the variable on its left-hand-side.

```
affine OneDTwoD {N,TMAX | N>0 && TMAX>0}
given
    float A {i| 0<=i<=N};
returns
    float B {t, i| 0<=i<=N && t==TMAX-1};
using
    float Local {t,i | 0<=i<=N && 0<=t<TMAX};
through
   Local[t,i] =
     case
       {|t>0 && i==0}: Local[t-1,i];
       {|t>0 && i==N}: Local[t-1,i];
       {|t>0 && 0<i<N}: 0.5 * (Local[t, i-1] + Local[t-1,i+1]);
     esac;
   B[t,i] = Local[t,i];
.
```

In the above example, variable 'Local' is not defined over {t==0 && i >= 0}. Check Program reports this error as:

```

```

```
ERROR::  Variable Local is not defined over the domain : {t,i|t== 0 && N-i>=
0 && TMAX-1>= 0 && N-1>= 0 && i>= 0}
```

Uniqueness Analysis

The second major property that Check Program tests is that at any point in the domain of a variable, there is a ***unique*** definition. This test manifests itself in many different ways, as described below.

Multiple Definitions for a Variable and Unused Variables

AlphaZ ensures that there is exactly one equation for each local and output variable. It also checks if any of the input or local variable is not used in the system.

```
affine OneDTwoD5 {N,TMAX | N>0 && TMAX>0}
given
    float A {i| 0<=i<=N};
returns
    float B {t, i| 0<=i<=N && t==TMAX-1};
using
    float Local {t,i | 0<=i<=N && 0<=t<TMAX};
through
   Local[t,i] =
     case
       {|t==0}: 0;
       {|t>0 && i==0}: Local[t-1,i];
       {|t>0 && i==N}: Local[t-1,i];
       {|t>0 && 0<i<N}: 0.5 * (Local[t, i-1] + Local[t-1,i+1]);
     esac;
   B[t,i] = Local[t,i];
   B[t,i] = Local[t,i];
.
```

In the above example, B is defined in multiple equations and variable 'A' is not used at all. Check program prints the following messages.

```
ERROR::  Variable B has multiple definitions (equations)
WARNING:: Variable A not used in any equation
```

Validity of the Case Statement

Ensures that multiple case subexpressions do not define same point in the domain of a variable.

```
affine OneDTwoD2 {N,TMAX | N>0 && TMAX>0}
given
    float A {i| 0<=i<=N};
returns
    float B {t, i| 0<=i<=N && t==TMAX-1};
using
    float Local {t,i | 0<=i<=N && 0<=t<TMAX};
through
   Local[t,i] =
     case
       {|t==0}: A[i];
       {|t>0 && i>=0}: Local[t-1,i];
       {|t>0 && i==N}: Local[t-1,i];
       {|t>0 && 0<i<N}: 0.5 * (Local[t, i-1] + Local[t-1,i+1]);
     esac;
   B[t,i] = Local[t,i];
.
```

In the above example, variable 'Local' is defined for {i > 0} in multiple case subexpressions. Check Program reports these errors as:

```
ERROR::  in the case statement : ({t,i|t-1>= 0} , {t,i|-N+i== 0 && t-1>= 0})
domains of subexpressions overlap on : {t,i|-N+i== 0 && t-1>= 0}
ERROR::  in the case statement : ({t,i|t-1>= 0} , {t,i|i-1>= 0 && t-1>= 0 &&
N-i-1>= 0}) domains of subexpressions overlap on : {t,i|i-1>= 0 && t-1>= 0
&& N-i-1>= 0}
```

Empty Sub Expressions

Expressions with empty domains can be the source of errors in the program.

```
affine OneDTwoD3 {N,TMAX | N>0 && TMAX>0}
given
    float A {i| 0<=i<=N};
returns
    float B {t, i| 0<=i<=N && t==TMAX-1};
using
    float Local {t,i | 0<=i<=N && 0<=t<TMAX};
through
   Local =
     case
       {t,i|t==0}: (t,i->N+1)@ A;
       {t,i|t>0 && i==0}: (t,i -> t-1,i)@Local;
       {t,i|t>0 && i==N}: (t,i -> t-1,i)@Local;
       {t,i|t>0 && 0<i<N}: 0.5 * ((t,i -> t, i-1)@Local + (t,i ->
t-1,i+1)@Local) + (t,i->N+1)@ A;
     esac;
```

```
   B[t,i] = Local[t,i];
.
```

In the above examples, domain of variable 'A' is {i| 0⇐i⇐N}, but in the equation for variable 'Local', cases {t,i|t==0} and {t,i|t>0 && 0<i<N} access A[N+1] which is not in the domain of variable 'A', thus resulting in expression with empty domain for both the cases.

```
WARNING::  This expression has empty domain : (t,i->N+1)@A
```

Parameter Related Analysis

If any local and output variables are not defined for any values of the parameters, then they are flagged as warnings.

```
affine OneDTwoD4 {N,TMAX | TMAX>0}
given
    float A {i| 0<=i<=N};
returns
    float B {t, i| 0<=i<=N && t==TMAX-1};
using
    float Local {t,i | 0<=i<=N && 0<=t<TMAX};
through
   Local[t,i] =
     case
       {|t==0}: A[i];
       {|t>0 && i==0}: Local[t-1,i];
       {|t>0 && i==N}: Local[t-1,i];
       {|t>0 && 0<i<N}: 0.5 * (Local[t, i-1] + Local[t-1,i+1]);
     esac;
   B[t,i] = Local[t,i];
.
```

In the above example, parameter 'N' has no restrictions, but none of the variables are declared for negative values of 'N'. Check Program reports these errors as:

```
WARNING::  Variable Local is not defined over the domain : {|TMAX-1>= 0 && -
N-1>= 0}
WARNING::  Variable B is not defined over the domain : {|TMAX-1>= 0 && -
N-1>= 0}
```

Not only declarations, but equations are also checked to ensure that they are defined for all values of parameters.