

Normalize

Sometimes, the expressions in an alphabets program is not written in the normal form as it is defined, which makes the program hard to read and understand. Nomalize is used to normalize the expressions in a program to the defined normal form. This page shows how to use Normalize to normalize a program.

Usage

The usage for the command Normalize is

```
Normalize(Program program);
```

The parameter program is the program you want to solve.

Normalization Rules

Normalize takes a program and applies a set of normalization rules on it. Some of the basic rules are shown below:

1. $\langle \text{latex} \rangle \$e.f \rightarrow e$, if $f(z)=z \langle \text{latex} \rangle$$
2. $\langle \text{latex} \rangle \$(e_1 \oplus e_2).f \rightarrow (e_1.f) \oplus (e_2.f) \langle \text{latex} \rangle$
3. $\langle \text{latex} \rangle \$(D:e_1) \oplus e_2 \rightarrow D:(e_1 \oplus e_2) \langle \text{latex} \rangle$
4. $\langle \text{latex} \rangle \$e_1 \oplus (D:e_2) \rightarrow D:(e_1 \oplus e_2) \langle \text{latex} \rangle$
5. $\langle \text{latex} \rangle \$(e \cdot f_1).f_2 \rightarrow e \cdot f$, where $f = f_1 \circ f_2 \langle \text{latex} \rangle$$
6. $\langle \text{latex} \rangle \$D_1:(D_2:e) \rightarrow D:e$, where $D=D_1 \cap D_2 \langle \text{latex} \rangle$$
7. $\langle \text{latex} \rangle \$(D:e).f \rightarrow D':e$, where $D' = f^{-1}(D) \langle \text{latex} \rangle$$

Example

Here we present an example and explain how the normalization rules work.

```
affine RestrictExpr {N | N > 1}
given
  int A {i | 0 <= i < 3N};
returns
  int C {i | 0 <= i < N};
  int D {i, j | 0 <= i < N && 0 <= j < N};
through
  C[i] = {i | 0 <= i < N}: ({i | 0 <= i < 2N}:A[i]);
  D[i,j] = (i, j -> j, i)@({i,j | 0 <= i < N}:A[i]);
.
```

In the above example, C simply copies the first N values of A, and $D[i,j] = A[i]$. Based on analysis, the normalization can be applied with the following code:

```
prog = ReadAlphabets("IdentityFunc.ab");
Normalize(prog);
```

The above code reads the program "IdentityFunc.ab" using command ReadAlphabets and applies normalization on the program. The following normalization rules can be applied to the program:

- In the computation for C, expression $A[i]$ is equivalent to $A.(i \rightarrow i)$, f is an identity function, rule number one is satisfied. So $A.(i \rightarrow i) \Rightarrow A$.
- In the computation for C, expression $\{i \mid 0 \leq i < N\} : (\{i \mid 0 \leq i < 2N\} : A[i])$ matches rule number 6, where $D1 = \{i \mid 0 \leq i < N\}$, $D2 = \{i \mid 0 \leq i < 2N\}$. $D = D1 \cap D2 = \{i \mid 0 \leq i < N\}$, the expression is changed to $\{i \mid 0 \leq i < N\} : A$.
- In the computation for D, expression $(i, j \rightarrow j, i) @ (\{i \mid 0 \leq i < N\} : A[i])$ matches rule number 7, where $D = \{i, j \mid 0 \leq i < N\}$, $f = (i, j \rightarrow j, i)$. $D' = f^{-1}(D) = \{i, j \mid 0 \leq j < N\}$, the expression is changed to $\{i, j \mid 0 \leq j < N\} : A[i, j]$;

The result for the above program after normalization is:

```
affine RestrictExpr {N | N > 1}
given
  int A {i | 0 <= i < 3N};
returns
  int C {i | 0 <= i < N};
  int D {i, j | 0 <= i < N && 0 <= j < N};
through
C = {i | 0 <= i < N} : A;
D = {i, j | 0 <= j < N} : A[i, j];
```

More Example

Here we give an example about Fibonacci:

```
affine Fib {N | N > 1}
given
returns
  int f;
using
  int fib {i | i >= 0 && -i >= -N};
through
  fib = case
    {i | i <= 1}: 1;
    {i | i >= 2}: (i -> i-1)@(case
      {i | i <= 1}: 1;
      {i | i >= 2}: (i -> i-1)@fib + (i -> i-2)@fib;
    esac)
  + (i -> i-2)@(case
    {i | i <= 1}: 1;
    {i | i >= 2}: (i -> i-1)@fib + (i -> i-2)@fib;
  esac);
  esac;
```

```
f = (->N)@fib;
```

```
.
```

In this example, the variable `fib` in the definition is substituted by its definition, which makes the program look complicated. We do normalization on the above program with the following commands:

```
prog = ReadAlphabets("Fib.ab");  
Normalize(prog);
```

The result program after normalization is

```
affine Fib {N | N > 1}  
given  
returns  
    int f;  
using  
    int fib {i | i >= 0 && -i >= -N};  
through  
fib = case  
    {i|-i+1>= 0} : (i->)@1;  
    {i|i-2== 0} : ((i->)@1 + (i->)@1);  
    {i|i-3== 0} : (((i->i-2)@fib + (i->i-3)@fib) + (i->)@1);  
    {i|i-4>= 0} : (((i->i-2)@fib + (i->i-3)@fib) + ((i->i-3)@fib +  
(i->i-4)@fib));  
    esac;  
f = (->N)@fib;  
.
```

It is clear that the normalized program is much easier to read and understand.

From:
<https://www.cs.colostate.edu/AlphaZ/wiki/> - **AlphaZ**

Permanent link:
<https://www.cs.colostate.edu/AlphaZ/wiki/doku.php?id=normalize&rev=1510669226>

Last update: **2017/11/14 07:20**

