High-Performance Embedded Systems-on-a-Chip

Lecture 11: Alpha (contd)

Sanjay Rajopadhye

Computer Science, Colorado State University

Outline

- Other Transformations:
  - Cut
  - Merge
  - Analyze
  - Simplify
  - AddLocal
- Array Notation
- Change of Basis

“Cut” the domain of X with $a^T z = a_0$

```
  ⋮
var    X : 𝒟_X of ...
  ⋮
 X = ⟨expr⟩
  ⋮
```

$$\begin{array}{l} \quad\vdots \\ \texttt{var}\quad \texttt{X} : \mathcal{D}_X\ \texttt{of} \ldots \\ \quad\vdots \\ \ \texttt{X} = \langle\texttt{expr}\rangle \\ \quad\vdots \end{array}$$

"Cut" the domain of X with $a^T z = a_0$

```
 ⋮
var      X : 𝒟_X of ...
 ⋮
 X = ⟨expr⟩
 ⋮
```

```
var      X : 𝒟_X of ...
var      X1 :  𝒟_X ∩ H of ...
var      X2 :  𝒟_X ∩ H′ of ...
 ⋮
 X1 = 𝒟_X ∩ H : ⟨expr⟩
 X2 = 𝒟_X ∩ H′ : ⟨expr⟩
 X = case X1; X2; esac;
 ⋮
```

where $H \equiv a^T z \geq a_0$, and $H' \equiv a^T z < a_0$ are the two halfspaces defined by $a^T z = a_0$.

“Cut” the domain of X with $a^T z = a_0$

```
  ⋮
var     X : 𝒟_X of ...
  ⋮
  X = ⟨expr⟩
  ⋮
```

```
var     X : 𝒟_X of ...
var     X1 :  𝒟_X ∩ H of ...
var     X2 :  𝒟_X ∩ H′ of ...
  ⋮
  X1 = 𝒟_X ∩ H : ⟨expr⟩
  X2 = 𝒟_X ∩ H′ : ⟨expr⟩
  X = case X1; X2; esac;
  ⋮
```

where $H \equiv a^T z \geq a_0$, and $H' \equiv a^T z < a_0$ are the two halfspaces defined by $a^T z = a_0$. Next, `substituteInDef` for ***all uses*** of `X`, and then eliminate `X`

Still More Transformations

- Cut
- Merge
- Analyze
- Simplify
- AddLocal

Array Notation

In general, a normalized equation has the form:

```
X = case
    ⟨Domain⟩ : ⟨VarOrConst⟩.⟨Dep⟩ op ...
        ⋮
    esac;
```

Array Notation

In general, a normalized equation has the form:

$$
\begin{array}{l}
\texttt{X = case} \\
\quad \langle\texttt{Domain}\rangle : \langle\texttt{VarOrConst}\rangle.\langle\texttt{Dep}\rangle \ \texttt{op} \ \ldots \\
\qquad \vdots \\
\quad \texttt{esac;}
\end{array}
$$

The number of index variables in

- any $\langle\texttt{Dep}\rangle$ to the left of the $\rightarrow$
- any $\langle\texttt{Domain}\rangle$ to the left of the |
- and in the $\langle\texttt{Domain}\rangle$ of $\texttt{X}$

Array Notation

In general, a normalized equation has the form:

```
X = case
    ⟨Domain⟩ : ⟨VarOrConst⟩.⟨Dep⟩ op ...
        ⋮
    esac;
```

The number of index variables in

- any ⟨Dep⟩ to the left of the $\rightarrow$
- any ⟨Domain⟩ to the left of the |
- and in the ⟨Domain⟩ of X

are all equal

Recipe

- rename all indices to be the same
- move them to the left of the equation
- drop them from the rhs (wherever they occur)

Recipe

- rename all indices to be the same
- move them to the left of the equation
- drop them from the rhs (wherever they occur)
- Add (syntactic) sugar, shake well and serve!

Outline

- Introduction
- ALPHA Syntax ALPHA
- (Denotational) Semantics
- Substitution, Normalization, & . . .
- Change of Basis (oh no, not again)

Consider the ALPHA program

$$\vdots$$

```
var    X : D_X of ...
  ⋮
  X = ⟨expr⟩
  ⋮
```

Consider the ALPHA program

$$\vdots$$

```
var    X : 𝒟_X of ...
  ⋮
 X = ⟨expr⟩
  ⋮
```

and affine functions $\mathcal{T}'$ and $\mathcal{T}$ such that $\mathcal{T}' \circ \mathcal{T} = \mathtt{I}$, i.e., $\mathcal{T}'(\mathcal{T}(z)) = z$

- Every occurrence of $\mathtt{X}$ (on the rhs of any equation) can be replaced by $\mathtt{X}.\mathcal{T}'.\mathcal{T}$, without affecting the semantics.
- Introduce a new variable

$$\mathtt{X}' = \mathtt{X}.\mathcal{T}' = \langle\mathtt{expr}\rangle.\mathcal{T}'$$

- Its domain is Pre$(\mathcal{D}_X, \mathcal{T}')$
- Replace every occurrence of the subexpression $\mathtt{X}.\mathcal{T}'$ in the program by $\mathtt{X}'$
- Since $\mathtt{X}$ is no longer used in the program, drop it, and then rename the $\mathtt{X}'$ to be $\mathtt{X}$

Summary: Three Simple Rules

Replace

- the domain $\mathcal{D}_X$ of $\mathtt{X}$ by $\mathsf{Pre}(\mathcal{D}_X, \mathcal{T})$
- all occurrences of $\mathtt{X}$ (on any rhs) by $\mathtt{X}.\mathcal{T}$
- compose a $\mathcal{T}'$ dependence at the end of the entire rhs of the equation of $\mathtt{X}$.

Example (Fibonacci again)

```
changeOfBasis["Fib.(i -> -i)", "i"]; show[]
```

Example (Fibonacci again)

```
        changeOfBasis["Fib.(i -> -i)", "i"]; show[]
system   FibSys :  {N| 1<=N} ()
returns  (F :      integer);
var      Fib :     {i | -N<=i<=-1} of integer;
let
Fib =    case
         {i|i<=2} :  1.(i->);
         {i|i>=3} :  Fib.(i->-i).(i->i-1)
                   + Fib.(i->-i).(i->i-2);
         esac.(i->-i);
F = Fib.(i->-i).( ->N);
tel;
```

Normalize again

```
                normalize[]; show[]
system   FibSys :  {N| 1<=N} ()
returns  (F :      integer);
var      Fib :     {i | -N<=i<=-1} of integer;
let
Fib =    case
         {i|-2<=i} :  1.(i->);
         {i|i<=-3} :  Fib.(i->i+1)
                   + Fib.(i->i+2)
         esac;
F = Fib.( ->-N);
tel;
```

Generalized CoB

- $\mathcal{T}$ is unimodular (simplest case)
- $\mathcal{T}$ is not square (but admits an integral left inverse),

Generalized CoB

- $\mathcal{T}$ is unimodular (simplest case)
- $\mathcal{T}$ is not square (but admits an integral left inverse), eg. alignment $(\mathtt{i} \rightarrow \mathtt{i}, \mathtt{i})$

Generalized CoB

- $\mathcal{T}$ is unimodular (simplest case)
- $\mathcal{T}$ is not square (but admits an integral left inverse), eg. alignment $(\mathtt{i} \to \mathtt{i}, \mathtt{i})$
- Can you do even better? What about a transformation $(\mathtt{i}, \mathtt{j} \to \mathtt{i}, \mathtt{0})$

Generalized CoB

- $\mathcal{T}$ is unimodular (simplest case)
- $\mathcal{T}$ is not square (but admits an integral left inverse), eg. alignment $(\mathtt{i} \rightarrow \mathtt{i}, \mathtt{i})$
- Can you do even better? What about a transformation $(\mathtt{i}, \mathtt{j} \rightarrow \mathtt{i}, \mathtt{0})$ applied to a variable whose domain is really a line-segment, "embedded" in a 2-D index space

- $\mathcal{T}$ is unimodular (simplest case)
- $\mathcal{T}$ is not square (but admits an integral left inverse), eg. alignment $(\mathtt{i} \rightarrow \mathtt{i}, \mathtt{i})$
- Can you do even better? What about a transformation $(\mathtt{i}, \mathtt{j} \rightarrow \mathtt{i}, \mathtt{0})$ applied to a variable whose domain is really a line-segment, "embedded" in a 2-D index space $\mathcal{T}$ must admit an integral left inverse, $\mathcal{T}'$ in the context of the variable $\mathtt{X}$, i.e.,

$$\forall z \in \mathcal{D}_X,\ \mathcal{T}'(\mathcal{T}(z)) = z$$

Why does it work?

- ALPHA domains (finite unions of polyhedra) constitute an abstract data type (ADT), closed under:
  - Intersection
  - Union (of finitely many members of the ADT)
  - Preimage by (arbitrary) affine functions (the class of dependences in ALPHA).
  - Image by unimodular affine functions (the class of transformations used for changes of bases).
- The class of transformations (unimodular affine functions) is a subset of the class of dependences
- the class of dependences is closed under composition