High-Performance Embedded Systems-on-a-Chip
Lecture 11: Alpha (contd)

Sanjay Rajopadhye

Computer Science, Colorado State University
Other Transformations:
  - Cut
  - Merge
  - Analyze
  - Simplify
  - AddLocal
Array Notation
Change of Basis
“Cut” the domain of $X$ with $a^T z = a_0$
“Cut” the domain of $X$ with $a^T z = a_0$

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

```plaintext
var x : $\mathcal{D}_X$ of ...

var x1 : $\mathcal{D}_X \cap H$ of ...

var x2 : $\mathcal{D}_X \cap H'$ of ...

... 

X1 = $\mathcal{D}_X \cap H : \langle expr \rangle$

X2 = $\mathcal{D}_X \cap H' : \langle expr \rangle$

X = case X1; X2; esac;
```

High-Performance Embedded Systems-on-a-Chip – p.3/14
“Cut” the domain of $X$ with $a^T z = a_0$

```plaintext
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 $X$. 

High-Performance Embedded Systems-on-a-Chip – p.3/14
Still More Transformations

- Cut
- Merge
- Analyze
- Simplify
- AddLocal
In general, a normalized equation has the form:

\[
X = \text{case} \\
\quad \langle \text{Domain} \rangle : \langle \text{VarOrConst} \rangle . \langle \text{Dep} \rangle \ op \ ... \\
\quad : \\
\quad \text{esac};
\]
In general, a normalized equation has the form:

\[
X = \text{case} \\
\quad \langle \text{Domain} \rangle : \langle \text{VarOrConst} \rangle . \langle \text{Dep} \rangle \ op \ldots \\
\quad \vdots \\
\quad \text{esac;}
\]

The number of index variables in

- any \( \langle \text{Dep} \rangle \) to the left of the →
- any \( \langle \text{Domain} \rangle \) to the left of the |
- and in the \( \langle \text{Domain} \rangle \) of \( X \)
In general, a normalized equation has the form:

\[ X = \text{case} \]
\[
\langle \text{Domain} \rangle : \langle \text{VarOrConst} \rangle . \langle \text{Dep} \rangle \ op \ ... \\
\quad \vdots \\
\quad \text{esac};
\]

The number of index variables in

- any \( \langle \text{Dep} \rangle \) to the left of the \( \rightarrow \)
- any \( \langle \text{Domain} \rangle \) to the left of the \( | \)
- and in the \( \langle \text{Domain} \rangle \) 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)
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

\[ \text{var } X : D_X \text{ of ...} \]

\[ \text{X = (expr)} \]
Consider the ALPHA program

\[ \text{var } X : D_X \text{ of } \ldots \]
\[ \vdots \]
\[ X = \langle \text{expr} \rangle \]
\[ \vdots \]

and affine functions \( \mathcal{T}' \) and \( \mathcal{T} \) such that \( \mathcal{T}' \circ \mathcal{T} = I \), i.e.,
\[ \mathcal{T}'(\mathcal{T}(z)) = z \]
• Every occurrence of $x$ (on the rhs of any equation) can be replaced by $x\cdot\mathcal{T}'\cdot\mathcal{T}$, without affecting the semantics.

• Introduce a new variable

$$x' = x\cdot\mathcal{T}' = \langle\text{expr}\rangle\cdot\mathcal{T}'$$

• Its domain is $\text{Pre}(\mathcal{D}_x, \mathcal{T}')$

• Replace every occurrence of the subexpression $x\cdot\mathcal{T}'$ in the program by $x'$

• Since $x$ is no longer used in the program, drop it, and then rename the $x'$ to be $x$
Summary: Three Simple Rules

Replace

- the domain $D_x$ of $x$ by $\text{Pre}(D_x, T)$
- all occurrences of $x$ (on any rhs) by $x.T$
- compose a $T'$ dependence at the end of the entire rhs of the equation of $x$. 
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;
system FibSys : \{N \mid 1 \leq N\} ()
returns (F : integer);
var Fib : \{i \mid -N \leq i \leq -1\} of integer;
let
Fib = case
{\{i\mid -2 \leq i\} : 1.(i->)};
{\{i\mid i \leq -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 ($i \rightarrow i, i$)
Generalized CoB

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

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

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