Colorado State University

Alpha, a Language for Parallel Hardware Arrays
Mauras 1989 PhD. thesis

Sanjay Rajopadhye
Colorado State University

Outline

- Introduction
- Alpha syntax
- (Denotational semantics)
  - Domains of expressions
  - Context domains

What is Alpha?

- Functional (equational) language
- Declarative
- Based on Systems of Afine Recurrence Equations on polyhedral domains
- with Reductions (& subsystems)

- A data-parallel language

The Essence of Data Parallelism

- Collections of elementary data objects
  - sets, bags, arrays
- Pointwise Pperations
- Alignment of different collections
- Conditional operations (on parts of) collections
- Reductions/Scans with associative/ commutative operators

Recap of Alpha (essence)

A program Name decls eqns consists of

- Name
  - system Name domain;
- Declarations: of Input (*I*), Output (*O*), Local (*L*) variable
  - type Vars domain;
- Equations
  - Var = Expr

Alpha syntax

- Domains: {IdxList | ConstrntList}
- Expressions:
  - Constant | Var V | C
  - Point-wise ops: Exp op Exp
  - Case: case Exp; … Exp; esac;
  - Restrict: Dom: Exp
  - Dependence: f@Exp | Exp.f
  - Reduce: reduce(op, f, Exp)

Alpha semantics (denotational)

- A system denotes a function from input variables to output variables
- An equation denotes a "rule" specifying how a variable is evaluated
- An expression denotes a mapping from indices (points in its domain) to values (i.e., multidimensional array/table of values)

Expression semantics (denotational)

Expressions are functions from domains to values. Semantics: two components: domain and a "meaning" function

- Domain: set of points where expression is defined.
  - "bounds checking" guarantees that NO out-of-bounds exception will be thrown
- Rules to construct domains are straightforward

Domains of Alpha Expressions

- Bottom up construction
  - Domain of a node in the AST is determined by that (those) of its child(ren)
- Simple rules involving properties of polyhedral sets
  - Intersection
  - Union
  - Image (by an affine function)
  - Pre-image (by an affine function)

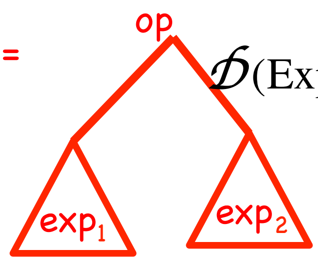
Constant/Variable (leaf nodes)

- Domain of a constant, `C` is $\mathcal{Z}^0$, the unique zero-dimensional polyhedron
- The domain of a variable `Var` is the domain of its declaration

Pointwise operators

- Domain of op(exp_1, exp_2)
  - $\mathcal{D}(exp)$ is the intersection of the domains of its children (same for k-ary operators)

Exp =

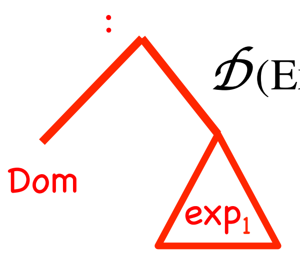


$$\mathcal{D}(\text{Exp}) = \mathcal{D}(\text{exp}_1) \cap \mathcal{D}(\text{exp}_2)$$

Restriction operator

- Domain of Dom:exp_1 is the intersection of the domains of its two components

Exp =

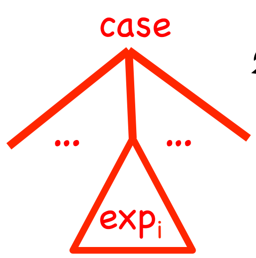


$$\mathcal{D}(\text{Exp}) = \text{Dom} \cap \mathcal{D}(\text{exp}_1)$$

Case operator

- Domain of `case … expi; … esac` is the disjoint union of the domains of its children

Exp =

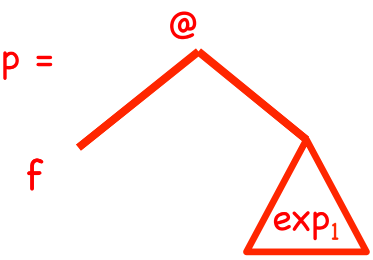


$$\mathcal{D}(\mathrm{Exp}) = \bigcup_{disj} \mathcal{D}(\mathrm{exp}_i)$$

Dependence

- Domain of `f@exp` is the pre-image of the domain of its child subexpression by the dep function

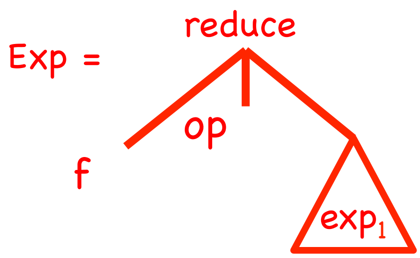
Exp =



$$\mathcal{D}(\mathrm{Exp}) = \mathrm{PreImage}(\mathcal{D}(\mathrm{exp}_1), f)$$

Reductions

- Domain of reduce(op, f, exp_1) is the image of the domain of its child subexpression by the function, f.



$$\mathcal{D}(\text{Exp}) = \text{Image}(\mathcal{D}(\text{exp}_1), f)$$

Dependences and Reductions

What does it mean and where is it defined?

- f@exp is an expression
  - whose value at z is the same as that of exp at f(z)
  - and domain is $f^{-1}(D_{expr})$
- reduce(op, f, exp) is an expression
  - whose domain is $f(D_{expr})$
  - and value at z is obtained by applying op to the values of exp at all points in $f^{-1}(z)$

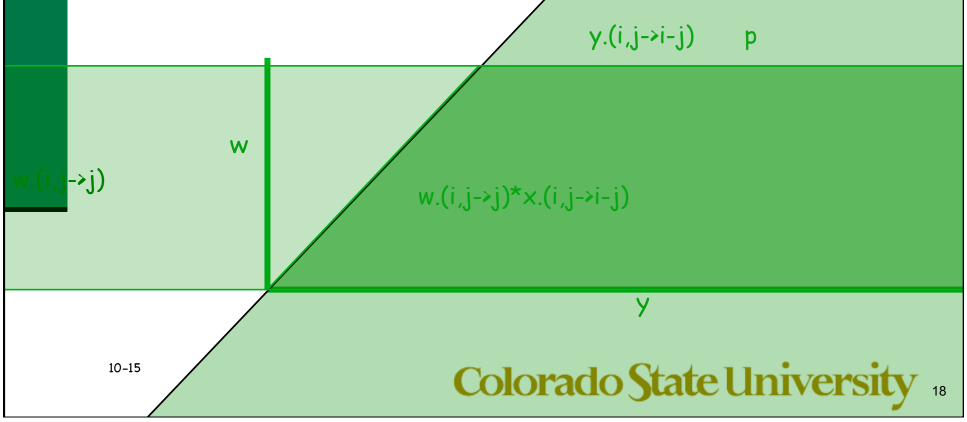
Example

```
affine fir {N|N>0}
input real w {i|N>=i>0};
output real y {i|i>=0};
through
 y = reduce(+, (i,j->i), (i,j->j)@w * (i,j->i-j)@y);
```

10-15 Colorado State University 17

Example domains

```
input real w {i|N>=i>0}; output real y {i|i>=0};
eqns: y = reduce(+, (i,j->i), w.(i,j->j) * y.(i,j->i-j));
```



10-15 Colorado State University 18

Context domain

- The context domain of a node in the AST is the set of points where it needs be evaluated in order to compute all the values of the LHS variable
- Deduced "top-down" from the parent's context domain
  - Based on the type of the parent node using grammar rules
    - Convention: exp_i is (usually) the node whose context domain we want, `exp` is the parent

Topmost node (equation)

- Expression whose parent node is an equation
  - Context domain of `exp` in the equation

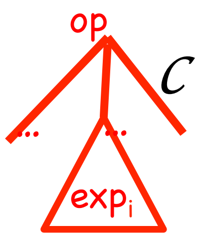
```
Var = exp;
```

is the domain of declaration of `Var`

Parent is pointwise operator

- op(…, exp_i, …)
  - The context domain of exp_i is the intersection of $\mathcal{C}$(Exp), and $\mathcal{D}(exp_i)$

Exp =



$$\mathcal{C}(\text{exp}_i) = \mathcal{D}(\text{exp}_1) \cap \mathcal{C}(\text{Exp}$$

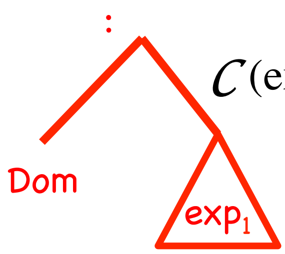
10-15

Colorado State University 21

Parent is restriction operator

- Dom: exp_1
  - Context domain of exp_1 is the intersection of the domains Dom, and $\mathcal{C}$(Exp)

Exp =



$$\mathcal{C}(\text{exp}_1) = \mathcal{C}(\text{Exp}) \cap \mathcal{D}om$$

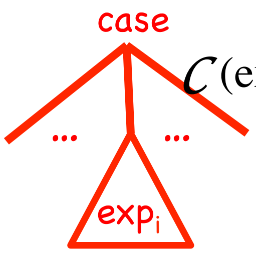
10-15

Colorado State University 22

Parent is case operator

- `case … exp`$_i$`; … esac`
  - Context domain of `exp`$_i$`;` is the intersection of the parent's context and its own domain

Exp =

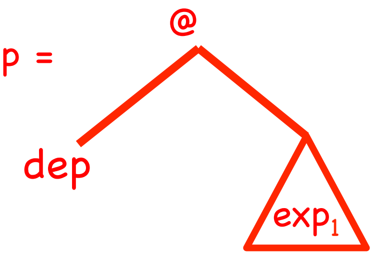


$$\mathcal{C}(\text{exp}_i) = \mathcal{D}(\text{exp}_i) \cap \mathcal{C}(\text{Exp})$$

Parent is a dependence

- `dep @ exp`
  - Context domain of `exp` is image of the context domain of its parent by dep

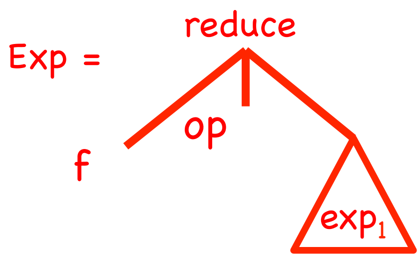
Exp =



$$\mathcal{C}(\text{exp}_1) = \text{Image}(\mathcal{C}(\text{Exp}), \text{dep}) \cap \mathcal{D}(\text{exp}_1)$$

Parent is a reduction

- reduce(op, f, exp_1)
  - Context domain of exp_1 is the pre-image of the context domain of its parent by the function, f.



$$\mathcal{C}(\exp_1) = \mathrm{PreImage}(\mathcal{C}(\mathrm{Exp}), f) \cap \mathcal{D}(\exp_1)$$