High-Performance Embedded Systems-on-a-Chip

Lecture 12: Executing Alpha

Sanjay Rajopadhye

Computer Science, Colorado State University

Outline

- Operational Semantics

Outline

- Operational Semantics
- Scanning polyhedra

Outline

- Operational Semantics
- Scanning polyhedra
- Horrendously inefficient code generation

Outline

- Operational Semantics
- Scanning polyhedra
- Horrendously inefficient code generation
- (In)efficient code: allocation memory for domains

Outline

- Operational Semantics
- Scanning polyhedra
- Horrendously inefficient code generation
- (In)efficient code: allocation memory for domains
- Efficient code generation

Operational Semantics of Alpha (expressions)

- Expressions denote mappings from indices to values
- Define a function Eval : $\langle \mathtt{Exp} \rangle \times \mathcal{Z}^n \rightarrow$ Type that actually (operationally) computes this mapping.

$$\mathsf{Eval}(\langle \mathtt{exp} \rangle, z) = \begin{cases} \mathsf{Eval}'(z) & \text{if } z \in \mathcal{D}(\mathtt{exp}) \\ \bot & \text{otherwise} \end{cases}$$

- Eval is defined recursively
- Six syntax rules $\Rightarrow$ six cases

The Eval′ function

- $\text{Eval}'(\langle \texttt{Const} \rangle, z) = C$

The Eval′ function

- $\text{Eval}'(\langle \texttt{Const} \rangle, z) = C$
- $\text{Eval}'(\langle \texttt{E1} \rangle \texttt{op} \langle \texttt{E2} \rangle, z) = \text{Eval}'(\langle \texttt{E1} \rangle, z) \oplus \text{Eval}'(\langle \texttt{E2} \rangle, z)$

The Eval′ function

- $\text{Eval}'(\langle \texttt{Const} \rangle, z) = C$
- $\text{Eval}'(\langle \texttt{E1} \rangle \texttt{op} \langle \texttt{E2} \rangle, z) = \text{Eval}'(\langle \texttt{E1} \rangle, z) \oplus \text{Eval}'(\langle \texttt{E2} \rangle, z)$
- $\text{Eval}'(\texttt{case..}\langle \texttt{Ei} \rangle \texttt{..esac}, z) =$

$$\begin{cases} \vdots & \\ \text{Eval}'(\langle \texttt{Ei} \rangle, z) & \text{if } z \in \mathcal{D}(\langle \texttt{Ei} \rangle) \\ \vdots & \end{cases}$$

The Eval′ function

- $\text{Eval}'(\langle \texttt{Const} \rangle, z) = C$
- $\text{Eval}'(\langle \texttt{E1} \rangle \texttt{op} \langle \texttt{E2} \rangle, z) = \text{Eval}'(\langle \texttt{E1} \rangle, z) \oplus \text{Eval}'(\langle \texttt{E2} \rangle, z)$
- $\text{Eval}'(\texttt{case..} \langle \texttt{Ei} \rangle \texttt{..esac}, z) =$

$$\begin{cases} \vdots & \\ \text{Eval}'(\langle \texttt{Ei} \rangle, z) & \text{if } z \in \mathcal{D}(\langle \texttt{Ei} \rangle) \\ \vdots & \end{cases}$$

- $\text{Eval}'(\texttt{D} : \langle \texttt{E} \rangle, z) = \text{Eval}'(\langle \texttt{E} \rangle, z)$

The Eval′ function

- $\text{Eval}'(\langle \texttt{Const} \rangle, z) = C$
- $\text{Eval}'(\langle \texttt{E1} \rangle \texttt{op} \langle \texttt{E2} \rangle, z) = \text{Eval}'(\langle \texttt{E1} \rangle, z) \oplus \text{Eval}'(\langle \texttt{E2} \rangle, z)$
- $\text{Eval}'(\texttt{case..}\langle \texttt{Ei} \rangle\texttt{..esac}, z) =$

$$\begin{cases} \vdots \\ \text{Eval}'(\langle \texttt{Ei} \rangle, z) \quad \text{if } z \in \mathcal{D}(\langle \texttt{Ei} \rangle) \\ \vdots \end{cases}$$

- $\text{Eval}'(\texttt{D} : \langle \texttt{E} \rangle, z) = \text{Eval}'(\langle \texttt{E} \rangle, z)$
- $\text{Eval}'(\langle \texttt{E} \rangle.f, z) = \text{Eval}'(\langle \texttt{E} \rangle, f(z))$

The Eval′ function

- $\text{Eval}'(\langle \texttt{Const} \rangle, z) = C$
- $\text{Eval}'(\langle \texttt{E1} \rangle \texttt{op} \langle \texttt{E2} \rangle, z) = \text{Eval}'(\langle \texttt{E1} \rangle, z) \oplus \text{Eval}'(\langle \texttt{E2} \rangle, z)$
- $\text{Eval}'(\texttt{case..}\langle \texttt{Ei} \rangle\texttt{..esac}, z) =$

$$\begin{cases} \vdots \\ \text{Eval}'(\langle \texttt{Ei} \rangle, z) \quad \text{if } z \in \mathcal{D}(\langle \texttt{Ei} \rangle) \\ \vdots \end{cases}$$

- $\text{Eval}'(\texttt{D} : \langle \texttt{E} \rangle, z) = \text{Eval}'(\langle \texttt{E} \rangle, z)$
- $\text{Eval}'(\langle \texttt{E} \rangle.f, z) = \text{Eval}'(\langle \texttt{E} \rangle, f(z))$
- $\text{Eval}'(\langle \texttt{Var} \rangle, z) = \text{EvalVar}(z)$

Semantics of Equations

- Equations do not denote mappings from indices to values

Semantics of Equations

- Equations do not denote mappings from indices to values
- Denotational Semantics: Equations denote "additions" to a "store of definitions"

Semantics of Equations

- Equations do not denote mappings from indices to values
- Denotational Semantics: Equations denote "additions" to a "store of definitions"
- Operational Semantics:

Semantics of Equations

- Equations do not denote mappings from indices to values
- Denotational Semantics: Equations denote "additions" to a "store of definitions"
- Operational Semantics:

$$\text{Eval}(\texttt{Var} = \langle \texttt{Exp} \rangle) = (\text{defun EvalVar}(\text{Eval}(\langle \texttt{Exp} \rangle z)$$

Semantics of programs (systems)

- Denotational Semantics: programs denote mappings from input variables to output variables
- (strict) Operational Semantics:

1. Read Input Variables
2. Compute (Local) and Output Variables
3. Write Output Variables

Scanning Polyhedra

- Given a polyhedron $\mathcal{P}$
- Problem: (generate code to) visit (in lexicographic order) all the integer points in $\mathcal{P}$

Nonstrict Operational Semantics

1. Read Input Variables
2. Write Output Variables (computing only those local variables that are necessary)

Nonstrict Operational Semantics

1. Read Input Variables
2. Write Output Variables (computing only those local variables that are necessary)

this will yield horribly inefficient code

Improvement

- Use memoization to avoid recomputation
- Allocate memory, store previously computed values, and evaluate only (at most) once

Drawbacks and Improvements

- Main Drawback: Too many context switches – factor of 5 to 8 for simple examples

Drawbacks and Improvements

- Main Drawback: Too many context switches – factor of 5 to 8 for simple examples
- Solution: Determine a schedule, and visit the points in the domains of the variables in that order

Drawbacks and Improvements

- Main Drawback: Too many context switches – factor of 5 to 8 for simple examples
- Solution: Determine a schedule, and visit the points in the domains of the variables in that order
- Key issues:
  - How to determine a schedule
  - How to exploit this to generate code (scanning unions of polyhedra)