# Denotational Semantics

A4M36TPJ, 2013/2014

# Operational Semantics

- Usually well suited for reasoning about whole programs, less than ideal for reasoning about program fragments.
- Sometimes tends to overspecify the implementation of certain language features (e.g. evaluation order).
- Tends to put emphasis on syntax (rather than semantics) of the language.

# Denotational Semantics

- Meaning of a program can be determined from the meaning of its parts.
- Unlike an operational semantics, a denotational semantics emphasizes ***what*** the meaning of a phrase is, not ***how*** the phrase is evaluated.

# Denotational Semantics

- Consists of three parts:
  - **Syntactic Algebra**
  - **Semantic Algebra**
  - **Meaning Function**

# Syntactic Algebra

- Describes abstract syntax of the language.
- Can be specified by a grammar.

# Semantic Algebra

- Models the meaning of program phrases.
- Consists of a collection of semantic domains along with functions that manipulate these domains.
- The meaning of a program may be as simple as an element of a primitive semantic domain like Int, the domain of integers.
- More typically, the meaning of a program is an element of a function domain that maps ***context domains*** to an ***answer domain***.

# Context Domains

- Denotational analogue of state components.
- Model such entities as name/value associations, contents of memory, and control information.

# Answer Domains

- Represent the possible meanings of programs.
- Include components that model context information that was transformed.

# Meaning Function

- Maps elements of the ***syntactic algebra*** (i.e., nodes in the abstract syntax trees) to their meanings in the ***semantic algebra***.
- Usually a collection of so-called ***valuation*** functions, one for each ***syntactic domain*** defined by the ***abstract syntax*** for the language.
- The function must be a **homomorphism** between the syntactic algebra and the semantic algebra.

# Meaning Function

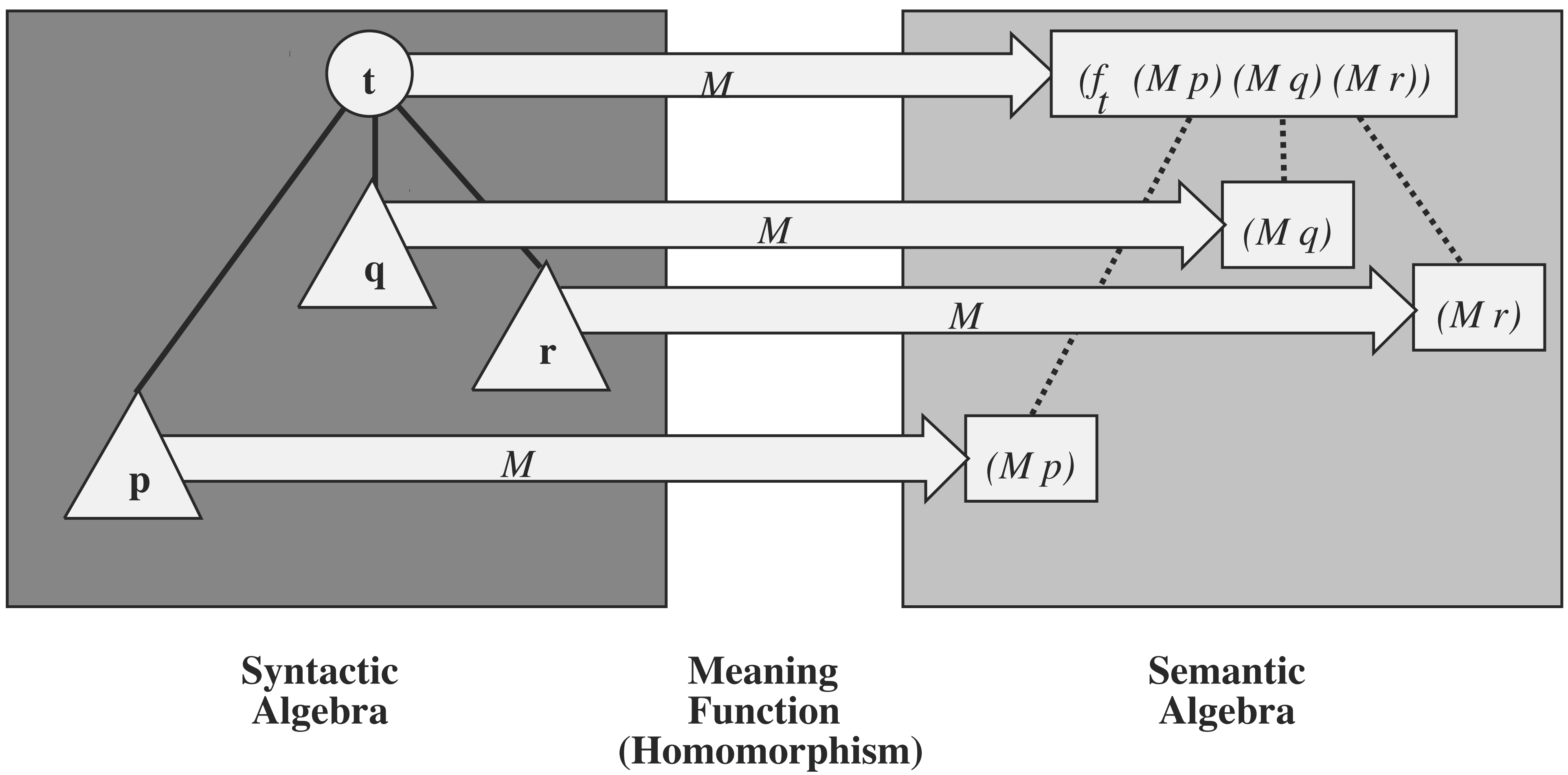
Suppose *M* is a meaning function and *t* is a node in an *abstract syntax tree*, with children $t_1, \ldots, t_k$. Then

$$(M\ t) = (f_t\ (M\ t_1)\ \ldots\ (M\ t_k))$$

where $f_t$ is a function that is determined by the syntactic class of t.

The reason to restrict meaning functions to homomorphisms is that their structure-preserving behavior greatly simplifies reasoning.

# Denotational Semantics Game



# Expression Language

# Syntax

$$
\begin{aligned}
Expr ::= \; & Num \mid \\
& \triangle Expr \mid \\
& Expr \odot Expr
\end{aligned}
$$

# Semantics

Semantic domain: $N$.

$$[\![n]\!] = n$$

$$[\![\triangle e]\!] = [\![\triangle]\!]([\![e]\!])$$

$$[\![\triangle]\!] = \lambda x. -x \text{ (i.e. unary minus)}$$

$$[\![e_1 \odot e_2]\!] = [\![\odot]\!]([\![e_1]\!], [\![e_2]\!])$$

$$[\![\odot]\!] = \lambda x, y. x + y \text{ (i.e. plus)}$$

# Logic Formulae Language

# Syntax

$$
\begin{aligned}
Formula ::= {} & true \mid \\
& false \mid \\
& \neg Formula \mid \\
& Formula\ BinaryConnective\ Formula \\
BinaryConnective ::= {} & \wedge \mid \vee
\end{aligned}
$$

# Semantics

Semantic domain: $\{0, 1\}$.

$$[\![true]\!] = 1$$

$$[\![false]\!] = 0$$

$$[\![\neg f]\!] = [\![\neg]\!]([\![f]\!])$$

$$[\![\neg]\!] = \lambda x.1 - x$$

# Semantics

$$[\![f_1 \ c \ f_2]\!] = [\![c]\!]([\![f_1]\!], [\![f_2]\!]) \quad (c \in BinaryConnective)$$

$$[\![\wedge]\!] = \lambda x, y. x \cdot y$$

$$[\![\vee]\!] = \lambda x, y. (x + y) - (x \cdot y)$$

# Regular Expressions

# Syntax

$$
\begin{aligned}
RegExp ::=\ & \emptyset \mid \\
& \epsilon \mid \\
& A \mid \\
& RegExp^* \mid \\
& RegExp\ BinOp\ RegExp \\
BinOp ::=\ & + \mid \cdot
\end{aligned}
$$

where $A$ is a predefined set of characters (alphabet).

# Semantics

Semantic domain: $A^*$.

$$[\![\emptyset]\!] = \{\}$$

$$[\![\epsilon]\!] = \{\epsilon\}$$

$$[\![a]\!] = \{a\}$$

$$[\![e^*]\!] = [\![^*]\!]([\![e]\!])$$

# Semantics

$$[\![*]\!] = \lambda L.\{l_1 \cdot \ldots \cdot l_n | n \in N \land l_i \in L\} \quad \text{(note: including } \epsilon\text{)}$$

$$[\![e_1 \ o \ e_2]\!] = [\![o]\!]([\![e_1]\!], [\![e_2]\!]) \quad (o \in BinOp)$$

$$[\![+]\!] = \cup$$

$$[\![\cdot]\!] = \lambda A, B.\{a \cdot b | a \in A \land b \in B\}$$

# Lambda Calculus

# Syntax

$$
\begin{aligned}
Expr ::= \; & X \mid \\
& \lambda X.Expr \mid \\
& Expr\ Expr
\end{aligned}
$$

# Semantics

Semantic domains: $env = string \to function$, $fcn = fcn \to fcn$; notational conventions $e \in env, f, f' \in fcn, E, E' \in Expr$

$$[\![x]\!] = \lambda e.e(x) \tag{26}$$

$$[\![\lambda x.E]\!] = \lambda e.\lambda p.[\![E]\!](e[x \mapsto p]) \tag{27}$$

$$[\![E_1 \ E_2]\!] = \lambda e.([\![E_1]\!](e))([\![E_2]\!](e)) \tag{28}$$