Attribute Semantics of Declarative Languages

Pavel Grigorenko

Institute of Cybernetics
Tallinn University of Technology

Voore Theory Days
29 September 2006

Outline

1. Attribute models
2. Evaluation of attributes
3. Higher-order attribute models
4. Evaluation of higher-order attributes
5. Languages

Introduction

- Declarative languages do not define a solution, they describe a problem
- Declarative specifications can describe single programs as well as artifacts
- A program can be obtained from a specification and a goal
- The meaning of a specification is a set of programs
- Implementation of this kind of semantics requires automation of program generation.

Introduction

- Declarative languages do not define a solution, they describe a problem
- Declarative specifications can describe single programs as well as artifacts
- A program can be obtained from a specification and a goal
- The meaning of a specification is a set of programs
- Implementation of this kind of semantics requires automation of program generation.

Introduction

- Declarative languages do not define a solution, they describe a problem
- Declarative specifications can describe single programs as well as artifacts
- A program can be obtained from a specification and a goal
- The meaning of a specification is a set of programs
- Implementation of this kind of semantics requires automation of program generation.

Introduction

- Declarative languages do not define a solution, they describe a problem
- Declarative specifications can describe single programs as well as artifacts
- A program can be obtained from a specification and a goal
- The meaning of a specification is a set of programs
- Implementation of this kind of semantics requires automation of program generation.

Introduction

- Declarative languages do not define a solution, they describe a problem
- Declarative specifications can describe single programs as well as artifacts
- A program can be obtained from a specification and a goal
- The meaning of a specification is a set of programs
- Implementation of this kind of semantics requires automation of program generation.

Attribute semantics – basic definitions

Attribute

Attribute is a variable with type

Functional dependency

$(y_1, \ldots, y_n) = f(x_1, \ldots, x_m)$

- $x_1, \ldots, x_m \rightarrow y_1, \ldots, y_n\{f\}$

Attribute dependency

Attribute dependency is a relation between attributes that is represented by one or several functional dependencies whose inputs and outputs are attributes bound by this relation

Attribute semantics – basic definitions

Attribute

Attribute is a variable with type

Functional dependency

$(y_1, \ldots, y_n) = f(x_1, \ldots, x_m)$

- $x_1, \ldots, x_m \rightarrow y_1, \ldots, y_n\{f\}$

Attribute dependency

Attribute dependency is a relation between attributes that is represented by one or several functional dependencies whose inputs and outputs are attributes bound by this relation

Attribute semantics – basic definitions

Attribute

Attribute is a variable with type

Functional dependency

$(y_1, \ldots, y_n) = f(x_1, \ldots, x_m)$

- $x_1, \ldots, x_m \rightarrow y_1, \ldots, y_n\{f\}$

Attribute dependency

Attribute dependency is a relation between attributes that is represented by one or several functional dependencies whose inputs and outputs are attributes bound by this relation

Attribute dependencies

Example

- Equality:
  $x = y$ can be rewritten as $x \rightarrow y;\ y \rightarrow x$
- Structural relation:
  $x = (x_1, ..., x_m)$ can be presented as $x_1, ..., x_m \rightarrow x;\ x \rightarrow x_1, ..., x_m$
- Equation:
  $x = y + z$ can be presented as a collection of functional dependencies, in the given example as $y, z \rightarrow x;\ x, y \rightarrow z;\ x, z \rightarrow y$
- Preprogrammed procedure:
  with attributes $x_1, ..., x_m$ as parameters producing a value of attribute y can be presented as $x_1, ..., x_m \rightarrow y$

Attribute dependencies

Example

- Equality:
  $x = y$ can be rewritten as $x \rightarrow y;\ y \rightarrow x$
- Structural relation:
  $x = (x_1, \ldots, x_m)$ can be presented as $x_1, \ldots, x_m \rightarrow x;\ x \rightarrow x_1, \ldots, x_m$
- Equation:
  $x = y + z$ can be presented as a collection of functional dependencies, in the given example as $y, z \rightarrow x;\ x, y \rightarrow z;\ x, z \rightarrow y$
- Preprogrammed procedure:
  with attributes $x_1, \ldots, x_m$ as parameters producing a value of attribute y can be presented as $x_1, \ldots, x_m \rightarrow y$

Attribute dependencies

Example

- Equality:
  $x = y$ can be rewritten as $x \to y;\ y \to x$
- Structural relation:
  $x = (x_1, \ldots, x_m)$ can be presented as $x_1, \ldots, x_m \to x;\ x \to x_1, \ldots, x_m$
- Equation:
  $x = y + z$ can be presented as a collection of functional dependencies, in the given example as $y, z \to x;\ x, y \to z;\ x, z \to y$
- Preprogrammed procedure:
  with attributes $x_1, \ldots, x_m$ as parameters producing a value of attribute y can be presented as $x_1, \ldots, x_m \to y$

Attribute dependencies

Example

- Equality:
  $x = y$ can be rewritten as $x \longrightarrow y;\ y \longrightarrow x$
- Structural relation:
  $x = (x_1, ..., x_m)$ can be presented as $x_1, ..., x_m \longrightarrow x;\ x \longrightarrow x_1, ..., x_m$
- Equation:
  $x = y + z$ can be presented as a collection of functional dependencies, in the given example as $y, z \longrightarrow x;\ x, y \longrightarrow z;\ x, z \longrightarrow y$
- Preprogrammed procedure:
  with attributes $x_1, ..., x_m$ as parameters producing a value of attribute y can be presented as $x_1, ..., x_m \longrightarrow y$

Attribute model

An *attribute model* M is a pair $\langle A, R\rangle$, where A is a finite set of attributes and R is a finite set of attribute dependencies binding these attributes

Composition of attribute models

- Attribute models $M^{'} = \langle A^{'}, R^{'}\rangle$ and $M^{''} = \langle A^{''}, R^{''}\rangle$
- Set of equalities $s = \{M^{'}.a = M^{''}.b, \dots \dots, M^{'}.d = M^{''}.e\}$
- Composition of $M^{'}$ and $M^{''}$: $\cup_s(M^{'}, M^{''})$, where $A^{'} \cup A^{''}$ and $R^{'} \cup R^{''} \cup s$
- Composition of attribute models: $\cup_s(M_1, \dots, M_n)$

Composite names

From $x,\ y \in m$ to $m.x,\ m.y$

Attribute models

Attribute model

An *attribute model* M is a pair $\langle A, R\rangle$, where A is a finite set of attributes and R is a finite set of attribute dependencies binding these attributes

Composition of attribute models

- Attribute models $M' = \langle A', R'\rangle$ and $M'' = \langle A'', R''\rangle$
- Set of equalities $s = \{M'.a = M''.b, \ldots \ldots, M'.d = M''.e\}$
- Composition of M' and M'': $\cup_s(M', M'')$, where $A' \cup A''$ and $R' \cup R'' \cup s$
- Composition of attribute models: $\cup_s(M_1, \ldots, M_n)$

Composite names

From $x, \ y \in m$ to $m.x, \ m.y$

Attribute models

Attribute model

An *attribute model* M is a pair $\langle A, R\rangle$, where A is a finite set of attributes and R is a finite set of attribute dependencies binding these attributes

Composition of attribute models

- Attribute models $M' = \langle A', R'\rangle$ and $M'' = \langle A'', R''\rangle$
- Set of equalities $s = \{M'.a = M''.b, \dots \dots, M'.d = M''.e\}$
- Composition of M' and M'': $\cup_s(M', M'')$, where $A' \cup A''$ and $R' \cup R'' \cup s$
- Composition of attribute models: $\cup_s(M_1, \dots, M_n)$

Composite names

From $x,\ y \in m$ to $m.x,\ m.y$

Attribute models contd.

Flattened form

Any attribute model can be represented in the flattened form (where each attribute dependency is a functional dependency). Relations between attributes are considered as sets of functional dependencies and their union is the set of attribute dependencies of the attribute model in the flattened form.

Example

Attribute model:

$M = \langle \{a; b; c; x; y; z\}, \{a = b + c; x = (y, z)\} \rangle$

Flattened form:

$M' = \langle \{a; b; c; x; y; z\}, \{b, c \rightarrow a;\ a, c \rightarrow b;\ a, b \rightarrow c;\ x \rightarrow y, z;\ y, z \rightarrow x\} \rangle$

Attribute models contd.

Flattened form

Any attribute model can be represented in the flattened form (where each attribute dependency is a functional dependency). Relations between attributes are considered as sets of functional dependencies and their union is the set of attribute dependencies of the attribute model in the flattened form.

Example

Attribute model:

$M = \langle\{a; b; c; x; y; z\}, \{a = b + c; x = (y, z)\}\rangle$

Flattened form:

$M' = \langle\{a; b; c; x; y; z\}, \{b, c \to a;\ a, c \to b;\ a, b \to c;\ x \to y, z;\ y, z \to x\}\rangle$

Computational problems of attribute models

Computational problem

- Let U and V be two sets of attributes of an attribute model M. We call a pair $\langle U, V \rangle$ a *computational problem* on the attribute model M, where U is a set of input attributes and V is a set of output attributes.
- *Given values of attributes from U find values of attributes of V using attribute dependencies of M*
- If for two computational problems $\langle U_1, V_1 \rangle$ and $\langle U_2, V_2 \rangle$ we have $U_1 \subseteq U_2$ and $V_2 \subseteq V_1$, and at least one of these inclusions is strict then we say that the computational problem $\langle U_1, V_1 \rangle$ is greater than the computational problem $\langle U_2, V_2 \rangle$.

Computational problems of attribute models

Computational problem

- Let U and V be two sets of attributes of an attribute model M. We call a pair $\langle U, V \rangle$ a *computational problem* on the attribute model M, where U is a set of input attributes and V is a set of output attributes.
- *Given values of attributes from U find values of attributes of V using attribute dependencies of M*
- If for two computational problems $\langle U_1, V_1 \rangle$ and $\langle U_2, V_2 \rangle$ we have $U_1 \subseteq U_2$ and $V_2 \subseteq V_1$, and at least one of these inclusions is strict then we say that the computational problem $\langle U_1, V_1 \rangle$ is greater than the computational problem $\langle U_2, V_2 \rangle$.

Computational problems of attribute models

Computational problem

- Let U and V be two sets of attributes of an attribute model M. We call a pair $\langle U, V \rangle$ a *computational problem* on the attribute model M, where U is a set of input attributes and V is a set of output attributes.
- *Given values of attributes from U find values of attributes of V using attribute dependencies of M*
- If for two computational problems $\langle U_1, V_1 \rangle$ and $\langle U_2, V_2 \rangle$ we have $U_1 \subseteq U_2$ and $V_2 \subseteq V_1$, and at least one of these inclusions is strict then we say that the computational problem $\langle U_1, V_1 \rangle$ is greater than the computational problem $\langle U_2, V_2 \rangle$.

Evaluation of attributes

Value propagation

Value propagation is a procedure that for an attribute model M in flattened form and a set of attributes U that belong to this model decides which attributes are computable from U and produces a sequence of functional dependencies that is an algorithm for computing values of these attributes

Example

$R = \{r = r1 + r2;\; u = i * r;\; u = u2 - u1\}$

$U = \{u1;\; u2;\; i\,\}$

$R' = \{r1, r2 \rightarrow r;\; r, r1 \rightarrow r2;\; r, r2 \rightarrow r1;\; i, r \rightarrow u;\; u, i \rightarrow r;\; u, r \rightarrow i;$
$u2, u1 \rightarrow u;\; u, u2 \rightarrow u1;\; u, u1 \rightarrow u2\}$

Algorithm:

- $\{u2, u1 \rightarrow u;\, \}$ $\quad U = \{u1;\; u2;\; i;\; u\}$
- $\{u2, u1 \rightarrow u;\; u, i \rightarrow r;\, \}$ $\quad U = \{u1;\; u2;\; i;\; u;\; r\}$

Evaluation of attributes

Value propagation

Value propagation is a procedure that for an attribute model M in flattened form and a set of attributes U that belong to this model decides which attributes are computable from U and produces a sequence of functional dependencies that is an algorithm for computing values of these attributes

Example

$R = \{r = r1 + r2;\ u = i * r;\ u = u2 - u1\}$
$U = \{u1;\ u2;\ i\ \}$
$R' = \{r1, r2 \rightarrow r;\ r, r1 \rightarrow r2;\ r, r2 \rightarrow r1;\ i, r \rightarrow u;\ u, i \rightarrow r;\ u, r \rightarrow i;$
$u2, u1 \rightarrow u;\ u, u2 \rightarrow u1;\ u, u1 \rightarrow u2\}$

Algorithm:

- $\{u2, u1 \rightarrow u;\ \}$ $U = \{u1;\ u2;\ i;\ u\}$
- $\{u2, u1 \rightarrow u;\ u, i \rightarrow r;\ \}$ $U = \{u1;\ u2;\ i;\ u;\ r\}$

Evaluation of attributes

Value propagation

Value propagation is a procedure that for an attribute model M in flattened form and a set of attributes U that belong to this model decides which attributes are computable from U and produces a sequence of functional dependencies that is an algorithm for computing values of these attributes

Example

$R = \{r = r1 + r2;\ u = i * r;\ u = u2 - u1\}$
$U = \{u1;\ u2;\ i\ \}$
$R' = \{r1, r2 \rightarrow r;\ r, r1 \rightarrow r2;\ r, r2 \rightarrow r1;\ i, r \rightarrow u;\ u, i \rightarrow r;\ u, r \rightarrow i;$
$\quad u2, u1 \rightarrow u;\ u, u2 \rightarrow u1;\ u, u1 \rightarrow u2\}$

Algorithm:

- $\{u2, u1 \rightarrow u;\ \}$ $\qquad U = \{u1;\ u2;\ i;\ u\}$
- $\{u2, u1 \rightarrow u;\ u, i \rightarrow r;\ \}$ $\qquad U = \{u1;\ u2;\ i;\ u;\ r\}$

Evaluation of attributes

Value propagation

Value propagation is a procedure that for an attribute model M in flattened form and a set of attributes U that belong to this model decides which attributes are computable from U and produces a sequence of functional dependencies that is an algorithm for computing values of these attributes

Example

$R = \{r = r1 + r2;\ u = i * r;\ u = u2 - u1\}$
$U = \{u1;\ u2;\ i\ \}$
$R^{'} = \{r1, r2 \rightarrow r;\ r, r1 \rightarrow r2;\ r, r2 \rightarrow r1;\ i, r \rightarrow u;\ u, i \rightarrow r;\ u, r \rightarrow i;$
$\quad u2, u1 \rightarrow u;\ u, u2 \rightarrow u1;\ u, u1 \rightarrow u2\}$

Algorithm:

- $\{u2, u1 \rightarrow u;\ \}$ $\quad U = \{u1;\ u2;\ i;\ u\}$
- $\{u2, u1 \rightarrow u;\ u, i \rightarrow r;\ \}$ $\quad U = \{u1;\ u2;\ i;\ u;\ r\}$

Higher-order attribute models

Higher-order functional dependency

- A is a set of attributes
- P is a set of computational problems
- *Higher-order functional dependency* (*hofd*) has inputs from $A \cup P$ and outputs from A.
- Inputs from P are called *subtasks*.

Example

$(u \to v), (s \to t), x \to y$

Higher-order attribute models

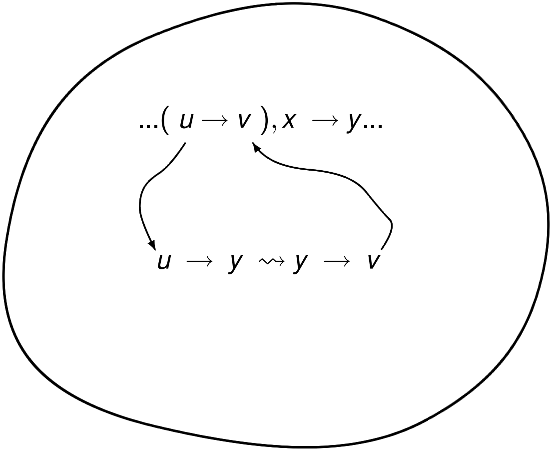
Higher-order functional dependency

- A is a set of attributes
- P is a set of computational problems
- *Higher-order functional dependency* (*hofd*) has inputs from $A \cup P$ and outputs from A.
- Inputs from P are called *subtasks*.

Example

$(u \to v),\ (s \to t),\ x \to y$

Higher-order functional dependency



Higher-order functional dependency

Example

$$z = \sum_{i=1}^{x} \sum_{j=1}^{y} a_{i,j}$$

$(\ i \rightarrow sum1\ ),\ x \rightarrow z$

$(\ j \rightarrow val\ ),\ y \rightarrow sum1$, where $sum1$ corresponds to $\sum_{j=1}^{y} a_j$

$i, j \rightarrow val$, where val corresponds to $a_{i,j}$

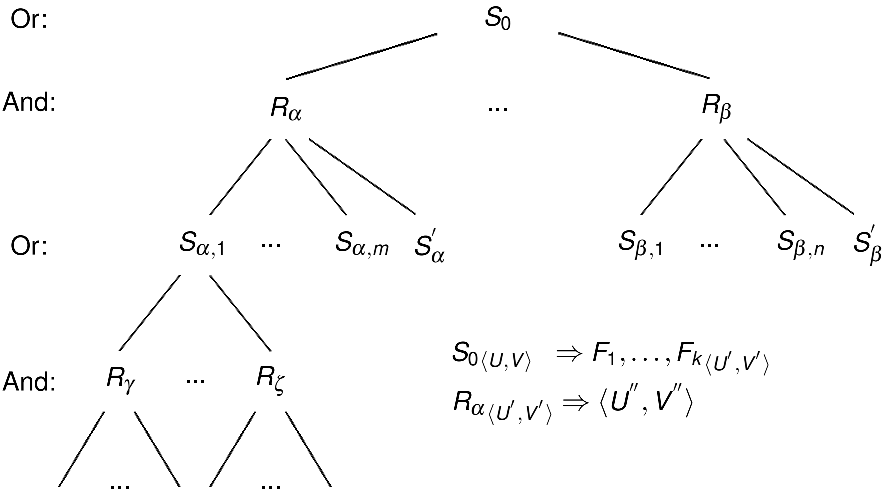
Evaluation of higher-order attributes

Maximal linear branch

- Value propagation produces: $\{F_1, \ldots, F_k\}$
- If the problem not solved, add *hofd*: $\{F_1, \ldots, F_k,\ R\}$
- *Maximal linear branch* (*mlb*) is a sequence of applicable functional dependencies with one *hofd* at the end.
  - A *mlb* cannot be found and the problem is unsolvable
  - The constructed *mlb* reduces the problem to a simpler one

And-or search tree

$R_i : \; S_{i,1}, \ldots, S_{i,m}, \underline{X} \rightarrow \underline{Y}$



$$S_{0\langle U,V\rangle} \Rightarrow F_1, \ldots, F_{k\langle U',V'\rangle}$$

$$R_{\alpha\langle U',V'\rangle} \Rightarrow \langle U'', V'' \rangle$$

Evaluation of higher-order attributes

Result of evaluation

- After constructing the *mlb* the problem is solvable (like in the case of a single *hofd*).
- A *mlb* cannot be found and the problem is unsolvable.
- A *mlb* can be found and the initial problem $\langle U_1, V_1 \rangle$ is reduced to a simpler one $\langle U_2, V_2 \rangle$, $U_2 = U_1 \cup Y$ and $V_2 = V_1 \setminus Y$, i.e. $\langle U_2, V_2 \rangle < \langle U_1, V_1 \rangle$, where Y is the set of outputs of the *hofd*.

The core language

The *core language* presents specifications as compositions of typed components. Each component has an attribute model that represents its semantic information. Attribute model of a component is determined by its type.

Statements:

- Declaration `<type> <identifier>;`
- Functional dependency (or *hofd*)
- Binding `<variable> = <variable>;`

Attribute model M of a specification is the composition of attribute models of its components $M' = \cup_s(M_1, \ldots, M_n)$ extended with attributes $a_1, \ldots, a_n$ declared by $D_1, \ldots, D_n$ and functional dependencies F.

Shallow semantics of the core language transforms specifications into attribute models.

Deep semantics of the core language

DS1

The deep semantics *DS1* of the core language produces an algorithm for any solvable computational problem on attribute model of a specification.

DS2

The deep semantics *DS2* produces an algorithm for solving the largest computational problem with an empty set of input attributes.

DS3

The real meaning of a specification can be computed as a value of a distinguished attribute on the attribute model of a specification.
The deep semantics *DS3* computes a value of such attribute for a given specification.

Deep semantics of the core language

DS1

The deep semantics *DS1* of the core language produces an algorithm for any solvable computational problem on attribute model of a specification.

DS2

The deep semantics *DS2* produces an algorithm for solving the largest computational problem with an empty set of input attributes.

DS3

The real meaning of a specification can be computed as a value of a distinguished attribute on the attribute model of a specification.
The deep semantics *DS3* computes a value of such attribute for a given specification.

Deep semantics of the core language

DS1

The deep semantics *DS1* of the core language produces an algorithm for any solvable computational problem on attribute model of a specification.

DS2

The deep semantics *DS2* produces an algorithm for solving the largest computational problem with an empty set of input attributes.

DS3

The real meaning of a specification can be computed as a value of a distinguished attribute on the attribute model of a specification.
The deep semantics *DS3* computes a value of such attribute for a given specification.

Extensions of the core language

Standard extension

New statements:

- Valuation `<variable> = <value>;`
- Alias `alias <identifier> = (<variable>,...);`
- Equation `<arithmetic expression> = <arithmetic expression>;`

Visual language for schemes

- Names of components and ports
- Connecting ports
- Assigning values

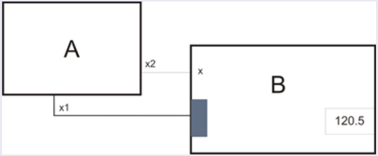
Extensions of the core language

Standard extension

New statements:

- Valuation `<variable> = <value>;`
- Alias `alias <identifier> = (<variable>,...);`
- Equation `<arithmetic expression> = <arithmetic expression>;`

Visual language for schemes



- Names of components and ports
- Connecting ports
- Assigning values

Attribute semantics of specifications

Steps for extracting the meaning of a specification

- translation into the core language
- translation from the core language into attribute model
- attribute evaluation

Summary

- Introduced a method for representing the semantics of specification languages by means of *attribute models*
- Presented the technique of *dynamic attribute evaluation* on simple attribute models as well as higher-order attribute models
- Defined three kinds of *deep semantics* of specifications
- *Attribute semantics* has been implemented in CoCoViLa *http://www.cs.ioc.ee/ ˜cocovila*