Semantic Hierarchy Refactoring by Abstract Interpretation

joint work with F. Logozzo, presented at VMCAI 2006
Overview

The goal A semantics-based framework for the definition and manipulation of class hierarchies for object-oriented languages.

The key-idea The notion of observable: an abstraction of the concrete semantics when focusing on a behavioral property of interest.

The main point We define a semantic subclass relation, capturing the fact that a subclass preserves the behavior of its superclass up to a given (tunable) observed property, and we study the relation between syntactic subclass and the notion of semantic subclass.

Instantiation We show algorithms that compute a semantic superclass for two given classes, that extend a hierarchy with a new class, and that merge two hierarchies by preserving semantic subclass relations.
Example: admissible hierarchies among classes

A hierarchy is admissible when the subclasses preserve a given property of their superclass.

Why do we need admissible class hierarchies? For two reasons:

- it allows one to design modular verification tools of polymorphic methods
- it supports the design of semantics-preserving operations on class hierarchies.
class Integer {
    int x;
    init(){ x = 0 }
    add() { x += 1 }
    sub() { x -= 1 }
}

class Even {
    int x;
    init(){ x = 0 }
    add() { x += 2 }
    sub() { x -= 2 }
}

class Odd {
    int x;
    init(){ x = 1 }
    add() { x += 2 }
    sub() { x -= 2 }
}

class MultEight{
    int x;
    init(){ x = 0 }
    add() { x += 16 }
    sub() { x -= 8 }
}

class MultTwelve{
    int x;
    init(){ x = 0 }
    add() { x += 24 }
    sub() { x -= 12 }
}
(a) $H_1$, admissible for congruences (and parities)

(b) $H_2$, admissible only for parities
Syntax

A **class** \( C \) is a triple \( \langle F, \text{init}, M \rangle \) where \( F \) is a set of distinct variables, \( \text{init} \) is the class constructor and \( M \) is a set of method definitions.

The set of all the classes is denoted by \( C \).

Given a class \( C = \langle F, \text{init}, M \rangle \), let \( M_{\text{names}} \) be the names of \( C \)'s methods. Then the **interface** of \( C \) is \( \iota(C) = F \cup \{\text{init}\} \cup M_{\text{names}} \).
Concrete Semantics of a method

Given a class $C = \langle F, \text{init}, M \rangle$, every instance of $C$ has an internal state $\sigma \in \Sigma$ that is a function from fields to values, i.e., $\Sigma = [F \to D_{val}]$, where $D_{val}$ is the semantic domain of values.

When a class is instantiated, the class constructor is called to set the internal state of the new object. This is modelled by a semantic function

\[
i[\text{init}] \in [D_{val} \to \mathcal{P}(\Sigma)].
\]

The **semantics of a method** $m$ is a function

\[
m[m] \in [D_{val} \times \Sigma \to \mathcal{P}(D_{val} \times \Sigma)].
\]

A method is called with two parameters: the method actual parameters and the internal state of the object it belongs to. The output of a method is a set of pairs $\langle \text{return value (if any), new object state} \rangle$. 
Class Reachable states

The most precise state-based property of a class $C$ is the set of states reached by any execution of every instance of $C$ in any possible context.

The set of states reached by any execution of any instance of a class can be expressed as a least fixpoint on the complete boolean lattice $\langle \mathcal{P}(\Sigma), \subseteq \rangle$.

The set of the initial states is:

$$S_0 = \{ \sigma \in \Sigma \mid \exists v \in D_{val}. \sigma \in i[init](v) \}.$$

The states reached after the invocation of a method $m$ are given by the method collecting forward semantics $\mathcal{M}^> [m] \in [\mathcal{P}(\Sigma) \rightarrow \mathcal{P}(\Sigma)]$:

$$\mathcal{M}^> [m](S) = \{ \sigma' \in \Sigma \mid \exists \sigma \in S. \exists v, v' \in D_{val}. \langle v', \sigma' \rangle \in m[m](v, \sigma) \}.$$
Class Reachable states

The class reachable states are the least solution of the following recursive equations:

\[ S' = S_0 \cup \bigcup_{m \in M} S_m \]

\[ S_m = \mathbb{M}^\succ [m] (S) \quad m \in M. \]  

The above equations characterize the set of states that are reachable before and after the invocation of any method in any instance of the class. Stated otherwise, they consider all the states reached after any possible invocation, in any order, with any input values of the methods of a class.
Methods’ preconditions

The method preconditions can be obtained by going backward from the postconditions: given a method \( m \) and its postcondition, we consider the set of states from which it is possible to reach a state in \( S_m \) by an invocation of \( m \). The collecting *backward* method semantics \( M^<[m] \in [\mathcal{P}(\Sigma) \rightarrow \mathcal{P}(\Sigma)] \) is defined as

\[
M^<[m](S) = \{ \sigma \in \Sigma \mid \exists \sigma' \in S. \exists v, v' \in D_{val}. \langle v', \sigma' \rangle \in m[m] \langle v, \sigma \rangle \}.
\]

and the methods preconditions are \( B_m = M^<[m](S_m) \).
Concrete Class Semantics

The concrete class semantics, i.e., the most precise property of a class, is the triple

$$\mathcal{C}[\mathcal{C}] = \langle S, S_0, \{m : B_m \rightarrow S_m \} \rangle$$
Domain of Observables

An observable of a class $C$ is an approximation of its semantics that captures some aspects of interest of the behavior of $C$. We build a domain of observables starting from an abstraction of sets of object states.

Let us consider an abstract domain $\langle P, \sqsubseteq \rangle$, which is a complete lattice, related to the concrete domain by a Galois connection:

$$\langle \mathcal{P}(\Sigma), \subseteq, \emptyset, \Sigma, \cup, \cap \rangle \xrightleftharpoons[\alpha]{\gamma} \langle P, \subseteq, \bot, \top, \sqcup, \sqcap \rangle.$$  \hspace{1cm} (2)

For instance, if we are interested in the linear relations between the values of the fields of the instances of $C$, we instantiate $P$ with the Octagons abstract domain. On the other hand if we are interested in object aliasing then we are likely to choose for $P$ an abstract domain that captures shapes.
Domain of observables of a class

Once $\langle P, \sqsubseteq \rangle$ is fixed, the abstract domain $\langle O[P], \sqsubseteq^K_o \rangle$ of the observables of a class is built on top of it. The elements of the abstract domain belong to the set:

$$O[P] = \{ \langle \bar{S}, S_0, \{ m : \langle V_m, B_m \rangle \rightarrow S_m \} \rangle | \bar{S}, S_0, V_m, B_m, S_m \in P\}.$$

Intuitively, an element of $O[P]$ consists of

- an approximation of the class invariant,

- the constructor postcondition, and

- for each method an approximation of its precondition and postcondition.

A method precondition has two parts, one for the method input values and the other for that internal object state.
The lattice $O[P]$

The order $\sqsubseteq_o$ is defined as:

$$o_1 \sqsubseteq_o o_2 \iff \bar{I} \subseteq \bar{J} \land \bar{I}_0 \subseteq \bar{J}_0 \land (\forall i. \bar{W}_i \subseteq \bar{U}_i \land \bar{Q}_i \subseteq \bar{R}_i \land \bar{I}_i \subseteq \bar{J}_i).$$

If $o_1$ and $o_2$ are the observables of two classes $A$ and $B$ then the order $\sqsubseteq_o$ ensures that $A$ preserves the class invariant of $B$ and that the methods of $A$ are a “safe” replacement of those with the same name in $B$.

**Theorem** Let $\langle P, \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$ be a complete lattice. Then $\langle O[P], \sqsubseteq_o, \bot_o, \top_o, \sqcup_o, \sqcap_o \rangle$ is a complete lattice. Moreover, if $\sqsubseteq$ is decidable then $\sqsubseteq_o$ is decidable too.
Lifting Galois connections to the domain of observables

\( \mathcal{A}(\mathcal{P}(\Sigma)) \), the set of all the abstractions of the concrete domain, is a complete lattice ordered w.r.t. the “relative” precision, \( \leq \), of abstract domains.

Let \( \langle P, \sqsubseteq \rangle \) and \( \langle P', \sqsubseteq' \rangle \) be two domains in \( \mathcal{A}(\mathcal{P}(\Sigma)) \) s.t. \( \langle P, \sqsubseteq \rangle \leq \langle P', \sqsubseteq' \rangle \) with the Galois connection \( \langle \alpha, \gamma \rangle \). Then,

\[
\langle \mathcal{O}[P], \sqsubseteq_o^P \rangle \leftrightarrow_{\alpha_o, \gamma_o} \langle \mathcal{O}[P'], \sqsubseteq_o^{P'} \rangle
\]

where \( \alpha_o \) and \( \gamma_o \) are

\[
\alpha_o(\langle \bar{S}, \bar{S}_0, \{m : \langle \bar{V}_m, \bar{B}_m \rangle \rightarrow \bar{S}_m \} \rangle) = \langle \alpha(\bar{S}), \alpha(\bar{S}_0), \{m : \langle \alpha(\bar{V}_m), \alpha(\bar{B}_m) \rangle \rightarrow \alpha(\bar{S}_m) \} \rangle
\]

\[
\gamma_o(\langle \bar{S}', \bar{S}_0', \{m : \langle \bar{V}_m', \bar{B}_m' \rangle \rightarrow \bar{S}_m' \} \rangle) = \langle \gamma(\bar{S}'), \gamma(\bar{S}_0'), \{m : \langle \gamma(\bar{V}_m'), \gamma(\bar{B}_m') \rangle \rightarrow \gamma(\bar{S}_m') \} \rangle.
\]
Abstract Semantics

An abstraction of $\mathbb{C} [\mathbb{C}]$ can be obtained by considering the abstract counterpart for the concrete semantics equation.

The best approximation for the initial states of the class is $\alpha(S_0) = \bar{S}_0$.

The best approximation in $P$ of the forward collecting method semantics of $m$ of $C$ is $\bar{M}^> [m] \in [P \to P]$ defined as $\bar{M}^> [m] (\bar{S}) = \alpha \circ \bar{M}^> [m] \circ \gamma(\bar{S})$.

The equation system becomes:

$$\bar{S} = \bar{S}_0 \sqcup \bigsqcup_{m \in M} \bar{S}_m$$

$$\bar{S}_m = \bar{M}^> [m] (\bar{S}) \quad m \in M.$$ (3)

The above equations are monotonic and, by the Tarski fixpoint theorem, there exists a least solution $\langle \bar{S}, \bar{S}_0, \{m : \bar{S}_m\} \rangle$.
The abstract preconditions can be obtained by considering the best approximation of the backward collecting method semantics $\overline{M}^<\llbracket m \rrbracket \in [P \rightarrow P]$ defined as

$$\overline{M}^<\llbracket m \rrbracket(\bar{S}) = \alpha \circ \overline{M}^<\llbracket m \rrbracket \circ \gamma(\bar{S})$$

The method abstract preconditions are obtained by projecting $\overline{M}^<\llbracket m \rrbracket(\bar{S}_m)$ respectively on the method input values and the instance fields:

$$\bar{V}_m = \pi_{in}(\overline{M}^<\llbracket m \rrbracket(\bar{S}_m)) \text{ and } \bar{B}_m = \pi_{F}(\overline{M}^<\llbracket m \rrbracket(\bar{S}_m)).$$

To sum up, the triple $\overline{C}[C] = \langle \bar{S}, \bar{S}_0, \{m : \langle \bar{V}_m, \bar{B}_m \rangle \rightarrow \bar{S}_m \} \rangle$ belongs to the domain of observables, and it is the best sound approximation of the semantics of $C$, w.r.t the properties encoded by the abstract domain $\langle P, \sqsubseteq \rangle$.

**Theorem** Let $\langle P, \sqsubseteq \rangle$ be an abstract domain and let the observable of a class $C$ w.r.t. the property encoded by $\langle P, \sqsubseteq \rangle$ be

$$\overline{C}[C] = \langle \bar{S}, \bar{S}_0, \{m : \langle \bar{V}_m, \bar{B}_m \rangle \rightarrow \bar{S}_m \} \rangle.$$ Then $\alpha_o(\overline{C}[C]) \sqsubseteq_o \overline{C}[C]$. 


Example

Let us instantiate $\langle P, \sqsubseteq \rangle$ with Con, the abstract domain of equalities of linear congruences. The elements of such a domain have the form $x = a \mod b$, where $x$ is a program variable and $a$ and $b$ are integers.

The representation function $\gamma_c \in [\text{Con} \rightarrow \mathcal{P}(\Sigma)]$ is defined as

$$
\gamma_c(x = a \mod b) = \{ \sigma \in \Sigma \mid \exists k \in \mathbb{N}. \sigma(x) = a + k \cdot b \}.
$$

Let us consider the classes Even and MultEight above, and let $e$ be the property $x = 0 \mod 2$, $d$ the property $x = 1 \mod 2$ and $u$ be the property $x = 0 \mod 8$.

Then the observables of Even and MultEight w.r.t. Con are

$$
\bar{C}[\text{Even}] = \langle e, e, \{ \text{add} : \langle \bot, e \rangle \rightarrow e, \text{sub} : \langle \bot, e \rangle \rightarrow e \} \rangle
$$

$$
\bar{C}[\text{Odd}] = \langle d, d, \{ \text{add} : \langle \bot, d \rangle \rightarrow d, \text{sub} : \langle \bot, d \rangle \rightarrow d \} \rangle
$$

$$
\bar{C}[\text{MultEight}] = \langle u, u, \{ \text{add} : \langle \bot, u \rangle \rightarrow u, \text{sub} : \langle \bot, u \rangle \rightarrow u \} \rangle.
$$
Syntactic Subclassing

The intuition behind the syntactic subclassing relation is inspired by the Smalltalk approach to inheritance: a subclass must answer to all the messages sent to its superclass. Stated otherwise, the syntactic subclassing relation is defined in terms of inclusion of class interfaces:

Let \( A \) and \( B \) be two classes, and the interface operator \( \iota(\cdot) \). Then the **syntactic subclass relation** is defined as \( A \triangleright B \iff \iota(A) \supseteq \iota(B) \).
Semantic Subclassing

Let \( \langle O, \sqsubseteq_o \rangle \) be an abstract domain of observables and let A and B be two classes. Then the **semantic subclassing relation** with respect to \( O \) is defined as \( A \triangleleft_O B \iff \complement C[A] \sqsubseteq_o \complement C[B] \).

The semantic subclassing relation formalizes the intuition that up-to a given property, a class A behaves like a class B.

For example, if the property of interest is the type of the class, then A is a semantic subclass of B if its type is a subtype of B.

In our framework, semantic subclassing can be defined in terms of the preservation of observables. In fact, as \( \sqsubseteq_o \) is the abstract counterpart for the logical implication then \( \complement C[A] \sqsubseteq_o \complement C[B] \) means that A preserves the semantics of B, when a given property of interest is observed.
Semantic subclass relation

When the abstract semantics of A and B are compared, that of A implies the one of B. This means that A refines B w.r.t. the properties encoded by the abstract domain $O$, in accord with the mundane approach of inheritance where a subclass is as a specialization of its ancestors.
Example

Let us consider the classes Even, Odd and MultEight and their respective observables wrt to the domain of congruences.

Then, as \( u \sqsubseteq e \) holds, we have that MultEight \( \triangleleft \) Even.

On the other hand, we have that neither \( e \sqsubseteq d \) nor \( d \sqsubseteq e \).

As a consequence, Even \( \not\triangleleft \) Odd and Odd \( \not\triangleleft \) Even.
Minimality

Let us consider two classes $A$ and $B$ that are equal except for a method $m$:

\[
\begin{align*}
A.m() &= \{ \\
& \quad x = 1; y = 2; \\
& \quad \text{if } (x > 0) \land (y \mod 2 = 0) \{ \\
& \quad \quad x = 1; y = 4; \\
& \quad \text{else } \{ \\
& \quad \quad x = 1; y = 8; \\
& \} \\
& \} \\
B.m() &= \{ \\
& \quad x = 1; y = 2; \\
& \quad \text{if } (x > 0) \land (y \mod 2 = 0) \{ \\
& \quad \quad x = 1; y = 2; \\
& \quad \text{else } \{ \\
& \quad \quad x = 3; y = 10; \\
& \} \\
& \} \\
\end{align*}
\]

$A \triangleleft_{\text{Intervals}} B$ as $(\[1, 1\],\[4, 8\]) \sqsubseteq (\[1, 3\],\[2, 10\])$

$A \triangleleft_{\text{Parities}} B$ as $(\text{odd, even}) \sqsubseteq (\text{odd, even})$.

In fact, in both cases the abstract domain is not precise enough to capture the branch chosen by the conditional statement.

Nevertheless, when considering the reduced product $\text{Intervals} \times \text{Parities}$ we have that $A \ntriangleleft_{\text{Intervals} \times \text{Parities}} B$ as

\[
((\[1, 1\],\text{odd}), (\[4, 4\],\text{even})) \not\subseteq ((\[1, 1\],\text{odd}), (\[2, 2\],\text{even})).
\]
Relationship between $\triangleright$ and $\triangleleft$

**Theorem** Let $A = \langle F_A, \text{init}_A, M_A \rangle$ and $B = \langle F_B, \text{init}_B, M_B \rangle$ be two classes such that $A \triangleright B$, and let $\langle P, \sqsubseteq \rangle \in \mathcal{A}(\mathcal{P}(\Sigma))$. If

(i) $I_B$ is a class invariant for $B$,

(ii) $\widetilde{M} \triangleright [\text{init}_A] \subseteq \widetilde{M} \triangleright [\text{init}_B],$

(iii) $\forall \bar{S} \in P. \forall m \in M_A \cap M_B. \widetilde{M} \triangleright [m](\bar{S}) \subseteq I_B$

(iv) $\forall m \in M_A. m \notin M_B \implies \widetilde{M} \triangleright [m](\bar{S}) \subseteq I_B$

then $A \triangleleft_{O[P]} B$.

**Theorem** Let $A, B \in \mathcal{C}$, such that $A \triangleleft_{O} B$. Then there exists a renaming function $\phi$ such that $\phi(A) \triangleright B$. 

Univ. Ca' Foscari di Venezia, march 02, 2006
Admissible Semantic Class Hierarchies

If $T$ is a tree, $\text{nodesOf}(T)$ denotes the elements of the tree, $\text{rootOf}(T)$ denotes the root of the tree, and if $n \in \text{nodesOf}(T)$ then $\text{sonsOf}(n)$ are the successors of the node $n$. In particular, if $\text{sonsOf}(n) = \emptyset$ then $n$ is a leaf. A tree with a root $r$ and successors $S$ is $\text{tree}(r, S)$.

Here we only consider single inheritance so that class hierarchies are trees of classes. An admissible hierarchy w.r.t. a transitive relation $\rho$ on classes is a tree such that all the nodes are classes, and given two nodes $n$ and $n'$ such that $n' \in \text{sonsOf}(n)$ then $n'$ is in the relation $\rho$ with $n$. Formally:

Let $\mathcal{H}$ be a tree and $\rho \subseteq \mathcal{C} \times \mathcal{C}$ be a transitive relation on classes. Then we say that $\mathcal{H}$ is a class hierarchy which is admissible w.r.t. $\rho$, if (i) $\text{nodesOf}(\mathcal{H}) \subseteq \mathcal{C}$, and (ii) $\forall n \in \text{nodesOf}(\mathcal{H}) \cdot \forall n' \in \text{sonsOf}(n) \cdot n' \rho n$.

We denote the set of all the class hierarchies admissible w.r.t. $\rho$ as $\mathcal{H}[\rho]$. 
Fair Operators

In order to manipulate hierarchies we wish to preserve admissibility. This is why we need the notion of a *fair* operator. A fair operator on class hierarchies transforms a set of class hierarchies admissible w.r.t. a relation $\rho$ into a class hierarchy that is admissible w.r.t. a relation $\rho'$.

Let $\rho$ and $\rho'$ be transitive relations. Then we say that a function $t$ is a *fair operator* w.r.t. $\rho$ and $\rho'$ if $t \in \mathcal{P}(\mathcal{H}[\rho]) \rightarrow \mathcal{H}[\rho']$.

In the following, when not stated otherwise, we assume that $\rho = \rho' = \triangleright$. 
Class Insertion

The first fair operator we consider inserts a class $C$ into an admissible class hierarchy $\mathcal{H}$. It uses a sub-routine that computes a common semantic superclass of two given classes. Four cases are distinguished.

- If $C$ already belongs to $\mathcal{H}$ then the hierarchy keeps unchanged.

- If $C$ is a superclass of the root of $\mathcal{H}$, then a new class hierarchy whose root is $C$ is returned.

- If $C$ is a subclass of the root of $\mathcal{H}$, then two subcases apply. If $C$ is a superclass of some of the successors, then it is inserted between the root of $\mathcal{H}$ and such successors. Otherwise it checks whether some root class of the successors is a superclass of $C$. If it is the case, then the algorithm is recursively applied, otherwise $C$ is added at this level of the hierarchy.

- If $C$ and the root of $\mathcal{H}$ are unrelated, the algorithm returns a new hierarchy whose root is a superclass of both $C$ and the root of $\mathcal{H}$.
\( \mathcal{H} \cup C \triangleq \) let \( R = \text{rootOf}(\mathcal{H}) \), \( S = \text{sonsOf}(\mathcal{R}) \)

let \( \mathcal{H}_< = \{ K \in S \mid \text{rootOf}(K) \lhd C \} \)

let \( \mathcal{H}_> = \{ K \in S \mid \text{rootOf}(K) \rhd C \} \)

if \( C \in \text{nodesOf}(\mathcal{H}) \) then return \( \mathcal{H} \)

if \( R \lhd C \) then return \( \text{tree}(C, R) \)

if \( C \lhd R \) then
  if \( \mathcal{H}_< \neq \emptyset \) then
    return \( \text{tree}(R, (S-\mathcal{H}_<) \cup \text{tree}(C, \mathcal{H}_<)) \)
  if \( \mathcal{H}_> \neq \emptyset \) then select \( K \in S \)
    return \( \text{tree}(R, (S-K) \cup (K \cup C)) \)
  else return \( \text{tree}(R, S \cup \{ C \}) \)

else select \( C_T = \text{CSS}(R, C) \)

return \( \text{tree}(C_T, \{ R, C \}) \)
Common Semantic Superclass

CSS(A, B) ≜ let A = ⟨F_A, init_A, M_A⟩,
B = ⟨F_B, init_B, M_B⟩,
F_C = ∅, init_C = init_A, M_C = ∅
repeat
    select f ∈ F_A − F_C
    if B ⊲⟨F_C ∪ {f}, τ_{F_C∪f}(init_A), τ_{F_C∪f}(M_C)⟩
        then F_C = F_C ∪ {f},
            init_C = τ_{F_C∪f}(init_A)
    ∥ select m ∈ M_A − M_C
    if B ⊲⟨F_C, init_C, τ_{F_C}(M_C ∪ {m})⟩
        then M_C = M_C ∪ {m}
until no more fields or methods are added
return ⟨F_C, init_C, τ_{F_C}(M_C)⟩
Merging of Hierarchies

The last refactoring operation on hierarchies we consider is about merging. The algorithm $\cup$ can be used as a basis for the algorithm to merge two admissible class hierarchies:

$$H_1 \cup H_2 \triangleq \text{let } H = H_1, N = \text{nodesOf}(H_2)$$

while $N \neq \emptyset$ do

select $C \in N$

$H = H \cup C$, $N = N - C$

return $H$. 
Conclusions and Future Work

We introduced a framework for the definition and the manipulation of class hierarchies based on semantics abstraction. The main novelty of this approach is twofold:

- it provides a logic-based solid foundation of class refactoring operations that are safe by construction, and

- it allows us to tune it according to the observed property.

The next goal is the development of a tool for the semi-automatic refactoring of class hierarchies, based on this work, and the design of abstract domains capturing properties expressible in JML.
Thanks!