# An Abstract Interpretation Framework for Refactoring

P. Cousot, *NYU, ENS, CNRS, INRIA*
R. Cousot, *ENS, CNRS, INRIA*
F. Logozzo, M. Barnett, *Microsoft Research*

## Example: extract method

```
public int Decrement(int x)
{
  Contract.Requires(x >= 5);
  Contract.Ensures(Contract.Result<int>() >= 0);

  while (x != 0) x--;

  return x;
}
```

```
public int Decrement(int x)
{
  Contract.Requires(x >= 5);
  Contract.Ensures(Contract.Result<int>() >= 0);

  x = NewMethod(x);

  return x;
}

private static int NewMethod(int x)
{
  while (x != 0) x--;

  return x;
}
```

## The problem

- Refactoring is a very common programmer activity
  - Useful to maintain the code, avoid code bloats, etc.
  - Examples: rename, re-order parameters, extract method, etc.
- IDEs guarantee that the refactored program is:
  1. a syntactically valid program
  2. a semantically equivalent program
- There is no guarantee about the
  1. Preservation of the correctness proof
  2. Interaction with the static analysis

## and the (modular) proof?

```
public int Decrement(int x)
{
  Contract.Requires(x >= 5);
  Contract.Ensures(Contract.Result<int>() >= 0);

  while (x != 0) x--;

  return x;
}
```

Postcondition: ok

No overlofw

```
public int Decrement(int x)
{
  Contract.Requires(x >= 5);
  Contract.Ensures(Contract.Result<int>() >= 0);

  x = NewMethod(x);

  return x;
}

private static int NewMethod(int x)
{
  while (x != 0) x--;

  return x;
}
```

Postcondition Violation?

Possible overlofw

## Simple solutions?

Method inlining: the reverse of extract method
- May not scale up, how many levels should we inline?

Isolated analysis: infer pre- and postconditions of the extracted method
- Too imprecise, without the context inferred contracts may be too generic

Invariant projection: project the pre/post-states on the parameters and return value
- Too specific, cannot refactor unreached code

User assistance: User provides the contracts
- Impractical, too many contracts to write
- State of the art (before this paper ;-)

# Extract method with contracts: Requirements

## Contribution

An abstract interpretation framework for proof-preserving method refactoring

A new set theoretic version of Hoare logic
- With some surprising results!

Definition of the problem of extract method with contracts
- Solution in the concrete and in the abstract

Implementation on a real system
- Using the CodeContracts static verifier (Clousot) and the Roslyn CTP
- Performance comparable to the "usual" extract method

## Validity

The inferred contract should be valid

Counterexample:

```
public int Decrement(int x)
{
 Contract.Requires(x >= 5);
 Contract.Ensures(Contract.Result<int>() >=0);

  x = NewMethod(x);

  return x;
}
```

ok

```
private static int NewMethod(int x)
{
  Contract.Requires(x >= 5);
  Contract.Ensures(Contract.Result<int>()==12345);

  while (x != 0) x--;

  return x;
}
```

Invalid ensures

# Safety

The precondition of the extracted method should advertise possible errors

Counterexample:

```
public int Decrement(int x)
{
 Contract.Requires(x >= 5);
 Contract.Ensures(Contract.Result<int>() >=0);

  x = NewMethod(x);

  return x;
}
```

ok

```
private static int NewMethod(int x)
{
  Contract.Ensures(Contract.Result<int>() == 0);

  while (x != 0) x--;

  return x;
}
```

Possible overflow

# Generality

The inferred contract is the most general satisfying Validity, Safety, and Completeness

Counterexample: Valid, Safe, Complete but not General contract

```
public int Decrement(int x)
{
 Contract.Requires(x >= 5);
 Contract.Ensures(Contract.Result<int>() >=0);

  x = NewMethod(x);

  return x;
}
```

ok

```
private static int NewMethod(int x)
{
  Contract.Requires(x >= 5);
  Contract.Ensures(Contract.Result<int>() == 0);

  while (x != 0) x--;

  return x;
}
```

Requires too strong

ok

# Completeness

The verification of the callee should still go through

Counterexample:  Valid and safe contract, but not complete

```
public int Decrement(int x)
{
 Contract.Requires(x >= 5);
 Contract.Ensures(Contract.Result<int>() >=0);

  x = NewMethod(x);

  return x;
}
```

Can't prove ensures

```
private static int NewMethod(int x)
{
  Contract.Requires(x >= 5);
  Contract.Ensures(Contract.Result<int>() <= x);

  while (x != 0) x--;

  return x;
}
```

ok

# Our solution

Valid, Safe, Complete, and General contract

```
public int Decrement(int x)
{
 Contract.Requires(x >= 5);
 Contract.Ensures(Contract.Result<int>() >=0);

  x = NewMethod(x);

  return x;
}
```

ok

```
private static int NewMethod(int x)
{
  Contract.Requires(x >= 0);
  Contract.Ensures(Contract.Result<int>() == 0);

  while (x != 0) x--;

  return x;
}
```

ok

# Formalization

## Orders on contracts

Covariant order $\Longrightarrow$

Intuition: a stronger precondition is better for the callee

$$P, Q \Longrightarrow P', Q' \text{ iff } P \subseteq P' \text{ and } Q \subseteq Q'$$

Controvariant order $\rightarrow$

Intuition: a $\rightarrow$-stronger contract is more general (better for the caller)

$$P, Q \rightarrow P', Q' \text{ iff } P' \subseteq P \text{ and } Q \subseteq Q'$$

*Note: formal (and more correct) definition in the paper*

## Algebraic Hoare Logic

We need to formalize what a static analyzer does, in particular method calls

Hoare Logic is the natural candidate

However, it is already an abstraction of the concrete semantics

We define a concrete Hoare logic where predicates are replaced by sets

$$\{ P \}\ S\ \{ Q \} \qquad P \in \wp(\Sigma) \text{ and } Q \in \wp(\Sigma \times \Sigma)$$

The deduction rules are as usual

Details in the paper

## Some notation...

`m` is the refactored (extracted) method

`S` denotes the selected code (to be extracted)

It is the body of the extracted method `m`

$P_m$, $Q_m$ is the most precise safety contract for a method m

*See Cousot, Cousot & Logozzo VMCAI'11*

$P_s$, $Q_s$ is the projection of the abstract state

before the selection, $P_s$

after the selection, $Q_s$

# Extract method with contracts problem

The refactored contract $P_R$, $Q_R$ is a solution to the problem if it satisfies

- Validity

$$\{ P_R \}\ S\ \{ Q_R \}$$

- Safety

$$P_R, Q_R \Longrightarrow P_m, Q_m$$

- Completeness

$$\{ P_s \}\ \mathtt{m(...)}\ \{ Q_s \}$$

- Generality

$\forall\ P'_R, Q'_R$ satisfying validity, safety, and completeness: $P_R, Q_R \rightarrow P'_R, Q'_R$

Theorem: The 4 requirements above are mutually independent

# Iterative Solution

Idea: give an iterative characterization of the declarative solution

- It is easier to abstract and compensates for the lose of precision

Theorem: Define

$$F[\mathbf{S}]\langle X, Y\rangle = \langle P_m \cap \text{pre}^{\sim}[\mathbf{S}]Y, Q_m \cap \text{post}[\mathbf{S}]X\rangle$$

Then

$$P_R, Q_R = \{ P_m \}\ \mathbf{S}\ \{ \text{post}[\mathbf{S}]P_m \} = \text{gfp}_{(Ps,\ Qs)}\ F[\mathbf{S}]$$

The order for the greatest fixpoint computation is $\rightarrow$

- Intuition: generalize the contract at each iteration step

# Declarative Solution

Theorem: There exists a unique solution for the problem:

$$P_R, Q_R = \{ P_m \}\ \mathbf{S}\ \{ \text{post}[\mathbf{S}]P_m \}$$

Drawback: It is not a feasible solution

- Pm and post[.] are not computable (only for trivial cases of finite domains)

We need to perform some abstraction to make it tractable

- The formulation above is ill-suited for abstraction

# Abstraction

# Abstract Hoare triples

Given abstract domains A approximating $\wp(\Sigma)$ and B approximating $\wp(\Sigma \times \Sigma)$

Define abstract Hoare triples

$$\{\, \underline{P} \,\}\ S\ \{\, \underline{Q} \,\} \iff \{\, \gamma_A(\underline{P}) \,\}\ S\ \{\, \gamma_B(\underline{Q}) \,\}$$

Idea: replace the concrete set operations with the abstract counterparts

Abstract Hoare triples generalize usual Hoare logic

- Example: Fix A, B to be first order logic predicates

Question: Are the usual rules of Hoare logic valid in the general case?

# We are in trouble?

A similar result holds for the disjunction rule ☹

We need some hypotheses on the abstract domains and the concretizations $\gamma$

Theorem: The abstract Hoare triples without the conjunction and disjunction are sound

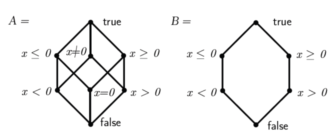
- But we need conjunction to model method call, product of analyses, etc.!

Theorem: If $\gamma_B$ is finite-meet preserving the conjunction rule is sound

A dual result holds for $\gamma_A$ and the disjunction rule

*Details on the paper: formalization and some extra technical details*

# Counterexample: conjunction rule




$$\{\, x \geq 0 \,\}\ \mathtt{x = -x}\ \{\, x \leq 0 \,\} \quad \text{and} \quad \{\, x \leq 0 \,\}\ \mathtt{x = -x}\ \{\, x \geq 0 \,\}$$

But

$$\{\, x \geq 0 \sqcap x \leq 0 \,\}\ \mathtt{x = -x}\ \{\, x \leq 0 \sqcap x \geq 0 \,\}$$

$$\{\, x = 0 \,\}\ \mathtt{x = -x}\ \{\, \text{false} \,\}$$

# And now?

We can define the problem of the extract method with contracts in the abstract

- Define abstract contracts, the rule for abstract method call, etc.

Theorem: The abstract counterparts for validity, safety, and completeness are sound

However, abstraction introduces new problems

- It is impossible to have a complete abstract refactoring in general
- It did not manifest in our experiments

The iterated gfp computation balances for the loss of information

Details in the paper (or come to see me after the talk!)

# Experiments

## Inference Algorithm

- Use the Roslyn refactoring service to detect the extracted method m
- Use Clousot to infer $P_s$, $Q_s$
  - Project the entry state on the beginning of the selection($P_s$). Similarly for $Q_s$
- Annotate the extracted method with $P_s$, $Q_s$
- Use Clousot to infer $P_m$, $Q_m$
- Add $P_m$, $Q_m$ to the extracted method and start the gfp computation
  - Weaken the precondition, strengthen the postcondition
  - Do not go below $P_s$, $Q_s$

## Implementation

- We use the CodeContracts static checker (aka Clousot) as underlying static analyzer
  - Based on abstract interpretation
  - More then 75K downloads, widely used in industrial environments
- We use the Roslyn CTP for C# language services and basic engine refactoring
  - Industrial strength C# compiler and services implementation
  - Integrates in Visual Studio

## Results

| Test | Extraction | Step 1 | Steps 2/3 | Total |
|---|---|---|---|---|
| Decrement | 0.18 | 0.10 | 0.12 | 0.42 |
| Generalize | 0.20 | 0.09 | 0.14 | 0.45 |
| BinarySearch | 0.23 | 0.14 | 0.32 | 0.70 |
| Abs | 0.23 | 0.07 | 0.12 | 0.43 |
| Arithmetic | 0.20 | 0.07 | 0.28 | 0.56 |
| Rem | 0.20 | 0.09 | 0.20 | 0.49 |
| Guard | 0.17 | 0.07 | 0.14 | 0.40 |
| Loop | 0.18 | 0.07 | 0.10 | 0.37 |
| Exp | 0.34 | 0.18 | 0.24 | 0.79 |
| Main | 0.20 | 0.14 | 0.20 | 0.56 |
| Karr | 0.35 | 0.09 | 0.14 | 0.71 |
| Loop-2 | 0.28 | 0.18 | 1.99 | 2.43 |
| Loop-3 | 0.21 | 0.10 | 0.14 | 0.46 |
| SankaEtAl [40] | 0.24 | 0.09 | 0.00 | 0.35 |
| McMillan [33] | 0.24 | 0.18 | 0.43 | 0.93 |
| BeyerEtAl [5] | 0.34 | 0.18 | 0.28 | 0.82 |
| PeronHalbwachs [28] | 0.47 | 0.33 | 0.31 | 1.13 |

# Conclusions

## Conclusions?

- Have an abstract interpretation framework to define proof-preserving refactorings
  - En passant, generalized Hoare logic
  - Found counterintuitive examples
- Instantiated to the problem of refactoring with contracts
  - In the concrete: One solution, two formulations
  - In the abstract: Completeness and generality only under some conditions
- Implementation on the top of industrial strength tools
- Come see our demo!!!