Computing Specifications

From Implementation to Specification by Construction.

André Platzer

University of Karlsruhe

Overview

- What means computing specifications
- Applications
- Feasibility
- How to do it
- Examples
- Summary & Discussion

Verification in reverse

- Given a program, automatically find its specification.
- Not just any specification, better yet the *strongest* specification that the program satisfies.

Applications

- Reverse-engineer existing code into a formal model.
- Auto-complete partial specifications of methods.
- Explain what a complex piece of code really does. (Expectation? Special cases?)

Terminology I

- ψ is a *specification* of a program α if

$$\models \; \mathtt{x} \doteq x^{\text{pre}} \; \rightarrow$$
$$[\alpha]\psi$$

Terminology I: Example

- ψ is a specification if

$$\models \; \mathtt{x} \doteq x^{\mathrm{pre}} \land \mathtt{y} \doteq y^{\mathrm{pre}} \;\rightarrow\; [\{\mathtt{int\ t} = \mathtt{x}; \mathtt{x} = \mathtt{y}; \mathtt{y} = \mathtt{t}; \}]\psi$$

Terminology I: Example

- ψ is a specification if

$$\models \; \mathtt{x} \doteq x^{\mathrm{pre}} \land \mathtt{y} \doteq y^{\mathrm{pre}} \;\rightarrow$$
$$[\{\mathtt{int\ t} = \mathtt{x}; \mathtt{x} = \mathtt{y}; \mathtt{y} = \mathtt{t};\}]\psi$$

- $\psi_1 \quad := \quad \mathtt{x} \doteq y^{\mathrm{pre}} \land \mathtt{y} \doteq x^{\mathrm{pre}} \; \surd$
- $\psi_2 \quad := \quad \mathtt{x} - \mathtt{y} \doteq y^{\mathrm{pre}} - x^{\mathrm{pre}} \; \surd$
- $\psi_3 \quad := \quad \mathtt{x} \neq x^{\mathrm{pre}} \land \mathtt{y} \neq y^{\mathrm{pre}} \; \times$

Terminology II

- ψ is *stronger* than ϕ:

$$\phi \preceq \psi \quad :\Longleftrightarrow \quad \psi \vDash \phi$$

Terminology II: Example

- ψ is *stronger* than ϕ:

$$\phi \preceq \psi \quad :\Longleftrightarrow \quad \psi \models \phi$$

- $\mathtt{x} - \mathtt{y} \doteq y^{\mathrm{pre}} - x^{\mathrm{pre}}$

$$\preceq \quad \mathtt{x} \doteq y^{\mathrm{pre}} \wedge \mathtt{y} \doteq x^{\mathrm{pre}}$$

Terminology II: Example

- ψ is *stronger* than ϕ:

$$\phi \preceq \psi \quad :\Longleftrightarrow \quad \psi \models \phi$$

- $\mathtt{x} - \mathtt{y} \doteq y^{\mathrm{pre}} - x^{\mathrm{pre}}$

$$\preceq \quad \mathtt{x} \doteq y^{\mathrm{pre}} \wedge \mathtt{y} \doteq x^{\mathrm{pre}}$$

- Do strongest specifications always exist?
  Are they unique?

Terminology II: Example

- ψ is *stronger* than ϕ:

$$\phi \preceq \psi \quad :\Longleftrightarrow \quad \psi \vDash \phi$$

- $\mathtt{x} - \mathtt{y} \doteq y^{\mathrm{pre}} - x^{\mathrm{pre}}$

$$\preceq \quad \mathtt{x} \doteq y^{\mathrm{pre}} \wedge \mathtt{y} \doteq x^{\mathrm{pre}}$$

- Do strongest specifications always exist? **Yes!**
  Are they unique? **Almost!**
- prefer "more readable" ψ.

Construction: Motivation

```
if (x > 9) {
    x = x + 2;
} else {
    x = x - 1;
}
x = x + 4;
```

What specification? Call it $\mathcal{C}$?

Construction: Motivation

```
if (x > 9) {
  x = x + 2;
} else {
  x = x - 1;
}
x = x + 4;
```

What specification? Call it $\mathcal{C}$?

$$\dfrac{\dfrac{\dfrac{\mathtt{x} > 9 \vdash \langle \mathtt{x} = \mathtt{x} + 6 \rangle \mathcal{C}}{\mathtt{x} > 9 \vdash \langle \mathtt{x} = \mathtt{x} + 2 + 4 \rangle \mathcal{C}}}{\mathtt{x} > 9 \vdash \langle \mathtt{x} = \mathtt{x} + 2 \rangle \langle \mathtt{x} = \mathtt{x} + 4 \rangle \mathcal{C}} \quad \dfrac{\dfrac{\mathtt{x} \leq 9 \vdash \langle \mathtt{x} = \mathtt{x} + 3 \rangle \mathcal{C}}{\mathtt{x} \leq 9 \vdash \langle \mathtt{x} = \mathtt{x} - 1 + 4 \rangle \mathcal{C}}}{\mathtt{x} \leq 9 \vdash \langle \mathtt{x} = \mathtt{x} - 1 \rangle \langle \mathtt{x} = \mathtt{x} + 4 \rangle \mathcal{C}}}{\vdash \langle \alpha \rangle \mathcal{C}}$$

Automatic construction

1. Call $\mathcal{C}$ a constant symbol of type formula.
2. Prepare (special) proof obligation.

$$\vdash \quad \mathtt{x} \doteq x^{\mathrm{pre}} \;\rightarrow\; \langle\alpha\rangle\mathcal{C}$$

3. Perform sequent calculus.
4. Use (failed) proof attempts.
5. Extract specification.

Inference rules

$$\frac{D_1 \quad ... \quad D_n}{C}$$

allowed if $\forall I \quad \forall s$

$$I, s \models C$$

$$\iff$$

$$I, s \models D_1 \land \ldots \land D_n$$

Ex: mutual branch

```
if (x < 0) {
    x = 0;
} else {
    x = x + 1;
}
if (0 > x) {
    x = 17;
} else {
    x = x + 2;}
```

Ex: mutual branch

- Implementation result (accumulator mode)

$$\psi_{\mathcal{C}} \quad := \quad (\mathtt{x} < 0 \rightarrow \langle \mathtt{x} = 2 \rangle \mathcal{C}) \\ \wedge (\mathtt{x} \geq 0 \rightarrow \langle \mathtt{x} = \mathtt{x} + 3 \rangle \mathcal{C})$$

Ex: mutual branch

- Implementation result (accumulator mode)

$$\psi_{\mathcal{C}} \quad := \quad (\mathtt{x} < 0 \rightarrow \langle \mathtt{x} = 2 \rangle \mathcal{C}) \\ \wedge (\mathtt{x} \geq 0 \rightarrow \langle \mathtt{x} = \mathtt{x} + 3 \rangle \mathcal{C})$$

- Implementation result (equation mode)

$$\psi' \quad := \quad (x^{\mathrm{pre}} < 0 \rightarrow x^{\mathrm{post}} \doteq 2) \\ \wedge (x^{\mathrm{pre}} \geq 0 \rightarrow x^{\mathrm{post}} \doteq x^{\mathrm{pre}} + 3)$$

Ex: algebraic loop

```
int i = 0;
while (i < 4) {
      s = s + n*i;
      i++;
}
```

- Implementation result (equation mode)

$$\psi_{\mathcal{C}} \quad := \quad s^{\text{post}} \doteq s^{\text{pre}} + \underbrace{6 * n}_{\text{simplified}} \wedge i^{\text{post}} \doteq 5$$

Ex: Diophantine equation

```
for (int i=1; i<10; i++) {
  for (int j=1; j<10; j++) {
    if (i*i + j + 1 == i*j) {
      // return "pair" (i,j)
      return (100 * i + j);
    }
  }
}
return -1;
```

Implementation

- As an additional KeY module.
- It works.
- Still some inconveniences: unnecessary internal variables.

Cons

- Specification collapses sometimes.
- Branching cases not (yet) reconciled.
- Unbounded loops are a major problem.

Pros

+ Apart from unbounded loops we have strongest specs.
+ No ad-hoc method but well-understood logic.
+ Close to theory.
+ Theorem provers are applicable.

Summary 1

- Computing specifications is useful.
- The strongest specification exists and is (rather) unique.
- Automatic specification construction is effective.
- Readability is the clue.

Summary 2

- Theoretical construction does not achieve readability.
- Theorem prover approach is better.
- Sequent calculus is applicable (in a natural way).
- Construction of logical representation $\neq$ inference of all program properties.

Repository

- The end of the presentation

Connexion

- ψ is a *specification* of program α if

$$\models \ \mathtt{x} \doteq x^{\mathrm{pre}} \ \rightarrow \ [\alpha]\psi(x)$$

$$\iff$$

$$\models \ \mathtt{x} \doteq x^{\mathrm{pre}} \ \rightarrow \ \left(\langle\alpha\rangle\mathtt{x} \doteq x^{\mathrm{post}} \ \rightarrow \ \psi\right)$$

Overall statement

$$\forall \mathcal{C} \quad \forall I \quad \forall s$$
$$I, s \models \ \mathtt{x} \doteq x^{\text{pre}} \ \rightarrow$$
$$(\langle \alpha \rangle \mathcal{C} \leftrightarrow \psi)$$

Especially

$$\mathcal{C} \quad := \quad \mathtt{x} \doteq x^{\text{post}}$$

Comparison with sp

- $sp(P, \ulcorner \mathrm{s} \urcorner) = Q \qquad \Rightarrow \qquad \vDash P \to \langle \mathbf{s} \rangle Q$

Comparison with sp

- $sp(P, \ulcorner \mathtt{s} \urcorner) = Q \qquad \Rightarrow \qquad \vDash P \rightarrow \langle \mathtt{s} \rangle Q$

- $$\frac{\Gamma, b \vdash \langle \mathtt{s} \rangle \phi, \Delta \quad \Gamma, \neg b \vdash \langle \mathtt{t} \rangle \phi, \Delta}{\Gamma \vdash \langle \mathtt{if(b)\ s\ else\ t} \rangle \phi, \Delta}$$

- $$sp(P, \ulcorner \textbf{if } (\mathtt{b}) \sim \mathtt{s} \sim \textbf{else} \sim \mathtt{t} \urcorner) = sp(P \wedge b, \ulcorner \mathtt{s} \urcorner) \vee sp(P \wedge \neg b, \ulcorner \mathtt{t} \urcorner)$$

Comparison with sp

- $sp(P, \ulcorner \mathtt{s} \urcorner) = Q \qquad \Rightarrow \qquad \vDash P \rightarrow \langle \mathtt{s} \rangle Q$

- $$\frac{\Gamma[\mathtt{x} \mapsto y], \mathtt{x} \doteq t[\mathtt{x} \mapsto y] \ \vdash \phi, \Delta[\mathtt{x} \mapsto y]}{\Gamma \ \vdash \langle \mathtt{x} := \mathtt{t} \rangle \phi, \Delta}$$

  where y is a new logical variable.

- $$sp(P, \ulcorner \mathtt{x := t} \urcorner) =$$
  $$\exists x_0 (P[x \mapsto x_0] \land x \doteq t[x \mapsto x_0])$$

Strongest specification ψ_α

- $\forall\alpha \ \exists\psi_\alpha$

$$\vDash \qquad \forall x^{\text{pre}} \ \forall x^{\text{post}}$$

$$\big(\mathtt{x} \doteq x^{\text{pre}} \rightarrow \ (\langle\alpha\rangle\mathtt{x} \doteq x^{\text{post}} \ \leftrightarrow \ \psi_\alpha)\big)$$

where $FV(\psi_\alpha) \subseteq \{x^{\text{pre}}, x^{\text{post}}\}$.

List of inference rules

"$+$"

$$\frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \land \psi, \Delta}$$

and-right

List of inference rules

“_”

$$\frac{\Gamma \vdash \Delta}{\Gamma, \phi \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \phi}$$

weakening

List of inference rules

"_"

$$\frac{\Gamma, \phi(X_1, \ldots, X_n, s(X_1, \ldots, X_n)) \ \vdash \Delta}{\Gamma, \exists y\, \phi(X_1, \ldots, X_n, y) \ \vdash \Delta}$$

Skolemise where $\{X_1, \ldots, X_n\}$ are the free variables occurring in $\exists y\, \phi(X_1, \ldots, X_n, y)$ and s is a new Skolem-function constant.

List of inference rules

“$+$”

$$\frac{\Gamma, \phi[y \mapsto \epsilon y\, \phi] \;\vdash \Delta}{\Gamma, \exists y\, \phi \;\vdash \Delta}$$

ϵ-rule, sometimes called “critical axiom”.

List of inference rules

"+"

$$\frac{\Gamma, \phi[x \mapsto t], \forall x\, \phi \ \vdash \ \Delta}{\Gamma, \forall x\, \phi \ \vdash \ \Delta}$$

universal quantifier

List of inference rules

"$+$"

$$\frac{\Gamma, b \vdash \langle \texttt{s} \rangle \phi, \Delta \quad \Gamma, \neg b \vdash \langle \texttt{t} \rangle \phi, \Delta}{\Gamma \vdash \langle \texttt{if}(\texttt{b})\ \texttt{s}\ \texttt{else}\ \texttt{t} \rangle \phi, \Delta}$$

branch

List of inference rules

"__"

$$\frac{\Gamma \ \vdash b, \Delta \quad \Gamma, b \ \vdash \langle \mathtt{s} \rangle \phi, \Delta}{\Gamma \ \vdash \langle \mathtt{if(b)\ s\ else\ t} \rangle \phi, \Delta}$$

"weakening" single-side branch

List of inference rules

"_"

$$\frac{\Gamma \vdash \phi \quad \Gamma, \phi \vdash \Delta}{\Gamma \vdash \Delta}$$

cut with weakening

List of inference rules

"+"

$$\frac{\Gamma \vdash \phi, \Delta \quad \Gamma, \phi \vdash \Delta}{\Gamma \vdash \Delta}$$

cut.

- But good idea?

List of inference rules

“$+$”

$$\frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \neg\phi, \Delta}{\Gamma \vdash \Delta}$$

cut-derived

List of inference rules

"$+$"

$$\frac{\Gamma \ \vdash \langle \texttt{if}(\texttt{b})\{\texttt{a}; \texttt{while}(\texttt{b})\texttt{a}\}\rangle\phi, \Delta}{\Gamma \ \vdash \langle \texttt{while}(\texttt{b})\texttt{a}\rangle\phi, \Delta}$$

unwind loop once

List of inference rules

“_”

$$\frac{\Gamma \vdash I \quad I, b \vdash [\mathtt{a}]I \quad I, \neg b \vdash \phi, \Delta}{\Gamma \vdash [\mathtt{while(b)a}]\phi, \Delta}$$

loop induction with invariant I.

List of inference rules

"_"

$$\frac{\Gamma[\mathtt{x} \mapsto y], \mathtt{x} \doteq t[\mathtt{x} \mapsto y] \ \vdash \phi, \Delta[\mathtt{x} \mapsto y]}{\Gamma \ \vdash \langle \mathtt{x} := \mathtt{t} \rangle \phi, \Delta}$$

assignment rule where y is a new logical variable (implicit universally quantified).

List of inference rules

"$+$"

$$\frac{\Gamma \ \vdash \phi[\mathtt{x} \mapsto t], \Delta}{\Gamma \ \vdash \langle \mathtt{x} := \mathtt{t} \rangle \phi, \Delta}$$

Example: unifyRobinson

```
==>
    (    t2 = self.IS_VARIABLE
       | t1 = self.IS_VARIABLE
       | {result:=1}
           ^true)
  & (    t2 + self.OCCURS_IN < t1
       & t2 = self.IS_VARIABLE
      -> t1 = self.IS_VARIABLE
       | {result:=0}
           ^true)
  & ...
```

Example: unifyRobinson

```
   ...
& (    t2 = self.IS_VARIABLE
    -> t2 + self.OCCURS_IN < t1
     | t1 = self.IS_VARIABLE
     | {result:=t2 + -t1}
         ^true)
& (    t1 + self.OCCURS_IN < t2
     & t1 = self.IS_VARIABLE
    -> {result:=0}
         ^true)
& ...
```

Example: unifyRobinson

```
   ...
& (   t1 = self.IS_VARIABLE
   -> t1 + self.OCCURS_IN < t2
    | {result:=t1 + -t2}
        ^true)
```

Example: mathPolyPuzzle

```
==>
  { i:=2,
    j:=5,
    result:=205}
    ^true
```

(Example: symbolic loop)

```
int loopy(int c, int a) {
   for (int i=0; i<3; i++) {
      c = c + a;
   }
   return c;
}
```

- Implementation result (equation mode)

$$\psi' \quad := \quad (\mathtt{result} \doteq \mathtt{c} + \underbrace{\mathtt{a} + \mathtt{a} + \mathtt{a}}_{3*\mathtt{a}})$$