Lecturecise 4
Refinement. Synthesis Procedures

Viktor Kuncak

2013

Local Variables

Global variables $V = \{x, z\}$
Program P:

$$x = x + 1; \{var\ \ y; y = x + 3; z = x + y + z\}; x = x + z$$

$R(P)$ should be a relation between (x, z) and (x', z').
Each statement should be relation between variables in scope

$$z = x + y + z$$

is relation between x, y, z and x', y', z'
Convention: consider the initial values of variables to be arbitrary
$R(y = x + 3; z = x + y + z) =$ $y' = x + 3 \wedge z' = 2x + 3 + z \wedge x' = x$

$R(\{var\ \ y; y = x + 3; z = x + y + z\}) =$ $\exists y, y'.\ y' = x+3 \wedge z' = 2x + 3 + z \wedge x' = x$

Local Variable Translation

R : programs → formulas

$R_V(P)$ is formula for P in the scope that has the set of variables P
For example,

$$R_V(x = t) \;=\; x' = t \wedge \bigwedge_{v \in V \setminus \{x\}} v' = v$$

Then define
$R_V(\{var\ y; P\}) \;=$ $\exists y', y.\ R_{V \cup \{y\}}(P)$

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables P
For example,

$$R_V(x = t) \;=\; x' = t \wedge \bigwedge_{v \in V \setminus \{x\}} v' = v$$

Then define
$R_V(\{var\ y; P\}) \;=\; \exists y. R_{V \cup \{y\}}(P)$

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables P
For example,

$$R_V(x = t) \; = \; x' = t \land \bigwedge_{v \in V \setminus \{x\}} v' = v$$

Then define
$R_V(\{var \;\; y; P\}) \; = \; \exists y. R_{V \cup \{y\}}(P)$

Exercise: express havoc(x) using var.

Local Variable Translation

$R_V(P)$ is formula for P in the scope that has the set of variables P
For example,

$$R_V(x = t) \;=\; x' = t \wedge \bigwedge_{v \in V \setminus \{x\}} v' = v$$

Then define
$R_V(\{var\ y; P\}) \;=\; \exists y.R_{V \cup \{y\}}(P)$

y'

Exercise: express havoc(x) using var.

$V = \{x, z\}$

$$R_V(havoc(x)) \iff R_V(\{var\ y;\ x = y\})$$

$z' = z$

$\exists y, \exists y'. (\ x' = y \wedge y' = y \wedge z' = z\ )$

$\exists y.\ \ x' = y \wedge z' = z$

$z' = z$

Havoc Multiple Variables at Once

Variables $V = \{x_1, \ldots, x_n\}$
Translation of $R(havoc(y_1, \ldots, y_m))$:

Havoc Multiple Variables at Once

Variables $V = \{x_1, \ldots, x_n\}$
Translation of $R(havoc(y_1, \ldots, y_m))$:

$$\bigwedge_{v \in V \setminus \{y_1, \ldots, y_m\}} v' = v$$

Exercise: the resulting formula is the same as for:

$$havoc(y_1); \ldots; havoc(y_n)$$

Programs and Specs are Relations

P	program:	$x = x + 2; y = x + 10$
$\rho(P)$	relation:	$\{(x, y, z, x', y', z') \mid x' = x + 2 \wedge y' = x + 12 \wedge z' = z\}$
$R(P)$	formula:	$x' = x + 2 \wedge y' = x + 12 \wedge z' = z$

Specification:

$$z' = z \wedge (x > 0 \rightarrow (x' > 0 \wedge y' > 0))$$

Adhering to specification is relation subset:

$$\begin{array}{rl} & \{(x, y, z, x', y', z') \mid x' = x + 2 \wedge y' = x + 12 \wedge z' = z\} \\ \subseteq & \{(x, y, z, x', y', z') \mid z' = z \wedge (x > 0 \rightarrow (x' > 0 \wedge y' > 0))\} \end{array}$$

Non-deterministic programs are a way of writing specifications

Writing Specs Using Havoc and Assume: Examples

Program variables $V = \{x, y, z\}$
Formula for relation (talks only about resulting state):

$$z' = z \land x' > 0 \land y' > 0$$

Corresponding program:

Writing Specs Using Havoc and Assume: Examples

Program variables $V = \{x, y, z\}$
Formula for relation (talks only about resulting state):

$$z' = z \land x' > 0 \land y' > 0$$

Corresponding program:

$$havoc(x, y); assume(x > 0 \land y > 0)$$

Writing Specs Using Havoc and Assume: Examples

Program variables $V = \{x, y, z\}$
Formula for relation (talks only about resulting state):

$$z' = z \land x' > 0 \land y' > 0$$

Corresponding program:

$$havoc(x, y); assume(x > 0 \land y > 0)$$

Formula for relation:

$$z' = z \land x' > x \land y' > y$$

Corresponding program?

Writing Specs Using Havoc and Assume: Examples

Program variables $V = \{x, y, z\}$
Formula for relation (talks only about resulting state):

$$z' = z \land x' > 0 \land y' > 0$$

Corresponding program:

$$havoc(x, y); assume(x > 0 \land y > 0)$$

Formula for relation:

$$z' = z \land x' > x \land y' > y$$

Corresponding program?
Use local variables to store initial values.

Writing Specs Using Havoc and Assume: Examples

Program variables $V = \{x, y, z\}$
Formula for relation (talks only about resulting state):

$$z' = z \wedge x' > 0 \wedge y' > 0$$

Corresponding program:

$$havoc(x, y); assume(x > 0 \wedge y > 0)$$

Formula for relation:

$$z' = z \wedge x' > x \wedge y' > y$$

Corresponding program?
Use local variables to store initial values.

```
{ var x0; var y0; var z0;
  x0 = x; y0 = y; z0 = z;
  havoc(x,y,z)
  assume(x > x0 && y > y0) && z = z0
}
```

Writing Specs Using Havoc and Assume

Global variables $V = \{x_1, \ldots, x_n\}$
Specification

$$F(x_1, \ldots, x_n, x_1', \ldots, x_n')$$

Becomes

Writing Specs Using Havoc and Assume

Global variables $V = \{x_1, \ldots, x_n\}$
Specification

$$F(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$$

Becomes

$$\begin{array}{l} \{\ \mathit{var}\ \ y_1, \ldots, y_n; \\ \quad y_1 = x_1; \ldots; y_n = x_n; \\ \quad \mathit{havoc}(x_1, \ldots, x_n); \\ \quad \mathit{assume}(F(y_1, \ldots, y_n, x_1, \ldots, x_n))\ \} \end{array}$$

uc

Program Refinement

For two programs, define $P_1 \sqsubseteq P_2$ iff

$$R(P_1) \rightarrow R(P_2)$$

is a valid formula. As usual, $P_2 \sqsupseteq P_1$ iff $P_1 \sqsubseteq P_2$.

- $P_1 \sqsubseteq P_2$ iff $\rho(P_1) \subseteq \rho(P_2)$

Define $P_1 \equiv P_2$ iff $P_1 \sqsubseteq P_2 \wedge P_2 \sqsubseteq P_1$

- $P_1 \equiv P_2$ iff $\rho(P_1) = \rho(P_2)$

Example for $V = \{x, y\}$

$$\{var\ x0; havoc(x); assume(x > x0)\} \sqsupseteq (x = x + 1)$$

Proof: Use R to compute formulas for both sides and simplify them.

Program Refinement

For two programs, define $P_1 \sqsubseteq P_2$ iff

$$R(P_1) \to R(P_2)$$

is a valid formula. As usual, $P_2 \sqsupseteq P_1$ iff $P_1 \sqsubseteq P_2$.

- $P_1 \sqsubseteq P_2$ iff $\rho(P_1) \subseteq \rho(P_2)$

Define $P_1 \equiv P_2$ iff $P_1 \sqsubseteq P_2 \wedge P_2 \sqsubseteq P_1$

- $P_1 \equiv P_2$ iff $\rho(P_1) = \rho(P_2)$

Example for $V = \{x, y\}$

$$\{var\ x0; havoc(x); assume(x > x0)\} \sqsupseteq (x = x + 1)$$

Proof: Use R to compute formulas for both sides and simplify them.

$$x' = x + 1 \to x' > x$$

Stepwise Refinement Methodology

Start form a possibly non-deterministic specification P_0
Refine the program until it becomes deterministic and efficiently executable.

$$P_0 \sqsupseteq P_1 \sqsupseteq \ldots \sqsupseteq P_n$$

Example:

$$\begin{array}{ll} & havoc(x); assume(x > 0); havoc(y); assume(x > y) \\ \sqsupseteq & havoc(x); assume(x > 0); y = x + 1 \\ \sqsupseteq & x = 42; y = x + 1 \\ \sqsupseteq & x = 42; y = 43 \end{array}$$

y > x

In the last step program equivalence holds as well

Monotonicity with Respect to Refinement

Theorem: if $P_1 \sqsubseteq P_2$ then $(P_1; P) \sqsubseteq (P_2; P)$
Theorem: if $P_1 \sqsubseteq P_2$ then $(P; P_1) \sqsubseteq (P; P_2)$
Theorem: if $P_1 \sqsubseteq P_2$ and $P_1' \sqsubseteq P_2'$ then

$$(\textit{if}\ \ (*) P_1\ \ \textit{else}\ \ P_1') \sqsubseteq (\textit{if}\ \ (*) P_2\ \ \textit{else}\ \ P_2')$$

Preserving Domain

It is not interesting program development step $P \sqsupseteq P'$ is P' is false, or is false for most inputs.

Example:

$$(havoc(x); assume(x + x = y)) \quad \sqsupseteq \quad (assume(y = 6); x = 3)$$

$x' + x' = y \land y' = y \quad \leftarrow \quad y' = y \land y' = 6 \quad \land \; x' = 3$

When doing refinement $P \sqsupseteq P'$, which ensures

$$R(P') \rightarrow R(P)$$

P

$dom(\rho(P))$

$dom(r) = \{x \mid \exists x'. (x, x') \in r\}$

we also wish to preserve the *domain* of the relation between $\bar{x}, \bar{x}'$

- if P has some execution from $\bar{x}$ ending in x'
- then P' should also have some execution, ending in some x'' (even if it has fewer choices)

$$(\exists \bar{x}'. R(P)) \rightarrow (\exists \bar{x}''. R(P'))$$

This is weaker than $R(P) \rightarrow R(P')$.

Definition: domain formula of P is the formula $\exists \bar{x}'. R(P)$

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) =$

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$(havoc(x); assume(x + x = y)) \quad \sqsupseteq \quad (assume(y = 6); x = 3)$$

- $R(P) = x' + x' = y \wedge y' = y$
- $R(P') =$

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y \land y' = y$
- $R(P') = x' = 3 \land y' = 6 \land y' = y$

Does $P \sqsupseteq P'$ really hold?

Now consider the right hand side:

- domain of P is

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold?

Now consider the right hand side:

- domain of P is $\exists x', y'. x' + x' = y \wedge y' = y$
- equivalent to:

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold?

Now consider the right hand side:

- domain of P is $\exists x', y'. x' + x' = y \wedge y' = y$
- equivalent to: $y\%2 = 0$
- domain of P is:

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold?

Now consider the right hand side:

- domain of P is $\exists x', y'. x' + x' = y \wedge y' = y$
- equivalent to: $y \% 2 = 0$
- domain of P' is: $\exists x', y'. x' = 3 \wedge y' = 6 \wedge y' = y$
- equivalent to:

Domains in the Example

Consider our example $P \sqsupseteq P'$

$$\big(havoc(x); assume(x + x = y)\big) \quad \sqsupseteq \quad \big(assume(y = 6); x = 3\big)$$

- $R(P) = x' + x' = y \wedge y' = y$
- $R(P') = x' = 3 \wedge y' = 6 \wedge y' = y$

Does $P \sqsupseteq P'$ really hold?

Now consider the right hand side:

- domain of P is $\exists x', y'. x' + x' = y \wedge y' = y$
- equivalent to: $y \% 2 = 0$
- domain of P is: $\exists x', y'. x' = 3 \wedge y' = 6 \wedge y' = y$
- equivalent to: $y = 6$

Does domain formula of P' imply the domain formula of P?

Preserving Domain: Exercise

$R(P) = x' + x' \geq y' \wedge y' = y$

domain: $\exists x', y'. R(P) \leftrightarrow \exists x'. \; 2x' \geq y$ true

Given P:

$$havoc(x); assume(x + x \geq y)$$

$x + x - 1 \geq y \rightarrow x + x \geq y$

Find P_1 and P_2 such that

- $P \sqsupseteq P_1 \sqsupseteq P_2$
- no two programs among P, P_1, P_2 are equivalent
- programs P, P_1 and P_2 have equivalent domains
- the relation described by P_2 is a partial function

P_1: havoc(x); assume($x \geq y$)

$2x = y$

$2x = y \rightarrow 2x \geq y$

P_1: havoc(x); assume($x + x - 1 \geq y$) $\subseteq P$

$P_2 \subseteq P_1$

$x = \max(0, y)$

$R(P_2)$: $y' = y \wedge (y \geq 0 \rightarrow x' = y) \wedge (y < 0 \rightarrow x' = 0)$

P_2:

```
if (y>=0)
  x=y
else x=0
```

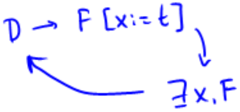
Complete Functional Synthesis

Domain-preserving refinement algorithm that produces a partial function

- assignment: **res** $=$ **choose x. F**
- corresponds to: $\{$**var x**; **assume**(**F**); **res** $=$ **x**$\}$
- we refine it preserving domain into: **assume**(**D**); **res** $=$ **t** (where t does not have 'choose')

More abstractly, given formula F and variable x find

- formula D
- term t not containing x



such that, for all free variables:

- $D \rightarrow F[x := t]$ (t is a term such that refinement holds)
- $D \iff \exists x.F$ (D is the domain, says when t is correct)

Consequence of the definition: $D \iff F[x := t]$

See Comfusy Examples on the Web

From Quantifier Elimination to Synthesis

Quantifier Elimination

If $\bar{y}$ is a tuple of variables not containing x, then

$$\exists x.(x = t(\bar{y}) \wedge F(x, \bar{y})) \iff F(t(\bar{y}), \bar{y})$$

Synthesis

$$\mathit{choose}\ x.(x = t(\bar{y}) \wedge F(x, \bar{y}))$$

gives:

- precondition $F(t(\bar{y}), \bar{y})$, as before, but also
- program that realizes x, in this case, $t(\bar{y})$

Handling Disjunctions

We had

$$\exists x.(F_1(x) \vee F_2(x))$$

is equivalent to

$$(\exists x.F_1(x)) \vee (\exists x.F_2(x))$$

Now:

$$\mathit{choose}\ x.(F_1(x) \vee F_2(x))$$

becomes:

$$\mathit{if}\ \ (D_1)\ (\mathit{choose}\ x.F_1(x))\ \mathit{else}\ (\mathit{choose}\ x.F_2(x))$$

where D_1 is the domain, equivalent to $\exists x.F_1(x)$ and computed while computing *choose* $x.F_1(x)$.

Framework for Synthesis Procedures

We define the framework as a transformation

- from specification formula F to
- the maximal domain D where the result x can be found, and the program t that computes the result

$\langle D \mid t \rangle$ denotes: the domain (formula) D and program (term) t
Main transformation relation $\vdash$

$$\mathit{choose}\ x.F \ \vdash \ \langle D \mid t \rangle$$

means

- $D \rightarrow F[x := t]$ (t is a term such that refinement holds)
- $D \iff \exists x.F$ (D is the domain, says when t is correct)

Rule for Synthesizing Conditionals

$$\frac{\mathit{choose}\ x.F_1 \vdash \langle D_1 \mid t_1 \rangle \quad \mathit{choose}\ x.F_2 \vdash \langle D_2 \mid t_2 \rangle}{\mathit{choose}\ x.(F_1 \lor F_2)\ \vdash\ \langle D_1 \lor D_2 \mid \mathit{if}\ (D_1)\ t_1\ \mathit{else}\ t_2 \rangle}$$

To synthesize the thing below the — , synthesize the things above and put the pieces together.

Test Terms Methods for Presburger Arithmetic Synthesis

Recall that the most complex step in QE for PA was replacing

$$\exists x. F_1(x)$$

with

$$\bigvee_{k=1}^{L} \bigvee_{i=1}^{N} F_1(a_k + i)$$

Now we transform *choose* $x.F_1(x)$ first into:

$$\textit{choose } x. \bigvee_{k=1}^{L} \bigvee_{i=1}^{N} (x = a_k + i \wedge F_1(x))$$

Then apply:

Test Terms Methods for Presburger Arithmetic Synthesis

Recall that the most complex step in QE for PA was replacing

$$\exists x.F_1(x)$$

with

$$\bigvee_{k=1}^{L} \bigvee_{i=1}^{N} F_1(a_k + i)$$

Now we transform *choose* $x.F_1(x)$ first into:

$$\textit{choose } x. \bigvee_{k=1}^{L} \bigvee_{i=1}^{N} (x = a_k + i \land F_1(x))$$

Then apply:

- rule for conditionals

Test Terms Methods for Presburger Arithmetic Synthesis

Recall that the most complex step in QE for PA was replacing

$$\exists x. F_1(x)$$

with

$$\bigvee_{k=1}^{L} \bigvee_{i=1}^{N} F_1(a_k + i)$$

Now we transform *choose* $x.F_1(x)$ first into:

$$\textit{choose } x. \bigvee_{k=1}^{L} \bigvee_{i=1}^{N} (x = a_k + i \land F_1(x))$$

Then apply:

- rule for conditionals
- one-point rule

Synthesis using Test Terms

$$choose\ x. \bigvee_{k=1}^{L} \bigvee_{i=1}^{N} (x = a_k + i \wedge F_1)$$

produces the same precondition as the result of QE, and the generated term is:

$$\begin{array}{l} \textit{if}\ \ (F_1[x := a_1 + 1])\ a_1 + 1 \\ \textit{elseif}\ \ (F_1[x := a_1 + 2])\ a_1 + 2 \\ \ldots \\ \textit{elseif}\ \ (F_1[x := a_k + i])\ a_k + i \\ \ldots \\ \textit{elseif}\ \ (F_1[x := a_L + N])\ a_L + N \end{array}$$

Linear search over the possible values, taking the first one that works.
This could be optimized in many cases–consider a project.

Synthesizing a Tuple of Outputs

$$\frac{\mathit{choose}\ x.F \ \vdash\ \langle D_1 \mid t_1 \rangle \quad \mathit{choose}\ y.D_1 \ \vdash\ \langle D_2 \mid t_2 \rangle}{\mathit{choose}\ (x,y).F \ \vdash\ \langle D_2 \mid (t_1[y := t_2],\ t_2) \rangle}$$

Note that y can appear inside D_1 and t_1, but not in D_2 or t_2

Automated Checks for Specifications: Uniqueness

Suppose we wish to give a warning if the specification F allows two different solutions.

Let the variables in scope be denoted by a and consider the synthesis problem:

$$choose\ x.\ F$$

What is the verification condition that checks whether the solution for x is unique?

Automated Checks for Specifications: Uniqueness

Suppose we wish to give a warning if the specification F allows two different solutions.

Let the variables in scope be denoted by a and consider the synthesis problem:

$$\textit{choose } x.\ F$$

What is the verification condition that checks whether the solution for x is unique?
Solution is **not** unique if this PA formula is satisfiable:

$$\exists a.\ \exists x_1, x_2.\ F[x := x_1] \land F[x := x_2] \land x_1 \neq x_2$$

Automated Checks for Specifications: Uniqueness

Suppose we wish to give a warning if the specification F allows two different solutions.

Let the variables in scope be denoted by a and consider the synthesis problem:

$$\mathit{choose}\ x.\ F$$

What is the verification condition that checks whether the solution for x is unique?
Solution is **not** unique if this PA formula is satisfiable:

$$F \wedge F[y := x] \wedge x \neq y$$

If we find such x, y, a we report them as an example that, for input a, there are two possible outputs, x and y

Automated Checks for Specifications: Totality

Suppose we wish to give a warning if in some cases the solution does not exist.

Let the variables in scope be denoted by *a* and consider the synthesis problem:

$$\textit{choose } x.\ F \quad \leadsto \quad \langle D \mid t \rangle$$

What is the verification condition that checks if there are cases when no solution *x* exists?

$$\forall a.\ \underbrace{\exists x.\ F}_{\equiv\ D}$$

$$\neg D$$

Automated Checks for Specifications: Totality

Suppose we wish to give a warning if in some cases the solution does not exist.

Let the variables in scope be denoted by a and consider the synthesis problem:

$$choose\ x.\ F$$

What is the verification condition that checks if there are cases when no solution x exists?
Check satisfiability of this PA formula:

$$\neg \exists x. F$$

If there is a solution a, report it as an example for which no solutions exist.

Further Topics

- demo
- handling equality and the consequence of Euclid's algorithm
- synthesis for sets with cardinality bounds