Complete Functional Synthesis

Definition (Synthesis Procedure)

A synthesis procedure takes as input formula F(x, a) and outputs a witness term Ψ such that F(Ψ,a) is a valid result of quantifier elimination.

That is, it outputs

1. a precondition formula *pre*(a)
2. list of terms Ψ

such that the following holds: $\exists x.F(x,a) \Leftrightarrow pre(a) \Leftrightarrow F[x := \Psi]$

Note: *pre(a)* is the “best” possible precondition under which a solution exists – domain of the relation.

choose((x, y) ⇒ 5 * x + 7 * y == a && x ≤ y)

Corresponding quantifier elimination problem:

$$\exists x \, \exists y \, . \, 5x + 7y = a \wedge x \le y$$

Use extended Euclid's algorithm to find particular solution to $5x + 7y = a$:

$x = 3a$
$y = -2a$

(5,7 are mutually prime, else we get divisibility pre.)

Express general solution of *equations* for x, y using a new variable z:

$x = -7z + 3a$
$y = 5z - 2a$

Rewrite *inequations* $x \le y$ in terms of z:

$5a \le 12z$

$\rightarrow z \ge \mathrm{ceil}(5a/12)$

Obtain synthesized program:

```
val z = ceil(5*a/12)
val x = -7*z + 3*a
val y = 5*z + -2*a
```

For a = 31:

z = ceil(5*31/12) = 13
x = -7*13 + 3*31 = 2
y = 5*13 – 2*31 = 3

Compilation in Comfusy

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  choose((h: Int, m: Int, s: Int) ⇒ (
        h * 3600 + m * 60 + s == totalSeconds
     && h ≥ 0
     && m ≥ 0 && m < 60
     && s ≥ 0 && s < 60    ))
```

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  val t1 =
  val t2 =
  val t3 =
  val t4 =
  (t1, t3, t4)
```

?

Mikaël Mayer   Ruzica Piskac   Philippe Suter, PLDI'10

Synthesis Procedure on Example

- process every equality: take an equality E_i, compute a parametric description of the solution set and insert those values in the rest of formula

$$\begin{pmatrix} h \\ m \\ s \end{pmatrix} = \lambda \begin{pmatrix} 1 \\ 0 \\ -3600 \end{pmatrix} + \mu \begin{pmatrix} 0 \\ 1 \\ -60 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ totalSeconds \end{pmatrix} \mid \lambda, \mu \in Z$$

CODE:
<further code will come here>
val h = lambda
val m = mu
val s = (-3600) * lambda + (-60) * mu + totalSeconds

Synthesis Procedure by Example

- process every equality: take an equality E_i, compute a parametric description of the solution set and insert those values in the rest of formula

$$\begin{pmatrix} h \\ m \\ s \end{pmatrix} = \lambda \begin{pmatrix} 1 \\ 0 \\ -3600 \end{pmatrix} + \mu \begin{pmatrix} 0 \\ 1 \\ -60 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ totalSeconds \end{pmatrix} \mid \lambda, \mu \in Z$$

Formula (remain specification):

0 ≤ λ, 0 ≤ μ, μ ≤ 59, 0 ≤ totalSeconds – 3600λ - 60μ, totalSeconds – 3600λ - 60μ ≤ 59

Processing Inequalities

process output variables one by one

$0 \leq \lambda$, $0 \leq \mu$, $\mu \leq 59$, $0 \leq \text{totalSeconds} - 3600\lambda - 60\mu$,
$\text{totalSeconds} - 3600\lambda - 60\mu \leq 59$

expressing constraints as bounds on μ

$0 \leq \lambda$, $0 \leq \mu$, $\mu \leq 59$, $\mu \leq \lfloor(\text{totalSeconds} - 3600\lambda)/60\rfloor$,
$\lceil(\text{totalSeconds} - 3600\lambda - 59)/60\rceil \leq \mu$

Code:

val mu = **min**(59, (totalSeconds -3600* lambda) **div** 60)

Fourier-Motzkin-Style Elimination

$0 \le \lambda$, $0 \le \mu$, $\mu \le 59$, $\mu \le \lfloor(\text{totalSeconds} - 3600\lambda)/60\rfloor$,
$\lceil(\text{totalSeconds} - 3600\lambda - 59)/60\rceil \le \mu$

combine each lower and upper bound

$0 \le \lambda$, $0 \le 59$, $0 \le \lfloor(\text{totalSeconds} - 3600\lambda)/60\rfloor$,
$\lceil(\text{totalSeconds} - 3600\lambda - 59)/60\rceil \le \lfloor(\text{totalSeconds} - 3600\lambda)/60\rfloor$,
$\lceil(\text{totalSeconds} - 3600\lambda - 59)/60\rceil \le 59$

basic simplifications

$0 \le \lambda$, $60\lambda \le \lfloor\text{totalSeconds}/60\rfloor$,
$\lceil(\text{totalSeconds} - 59)/60\rceil - 59 \le 60\lambda$

Code:

val lambda = totalSeconds **div** 3600

Preconditions: $0 \le \text{totalSeconds}$

Compilation in Comfusy

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  choose((h: Int, m: Int, s: Int) ⇒ (
          h * 3600 + m * 60 + s == totalSeconds
      && h ≥ 0
      && m ≥ 0 && m < 60
      && s ≥ 0 && s < 60    ))
```

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  val t1 = totalSeconds div 3600
  val t2 = totalSeconds -3600 * t1
  val t3 = t2 div 60
  val t4 = totalSeconds - 3600 * t1 - 60 * t3
  (t1, t3, t4)
```

Handling of Inequalities

- Solve for one by one variable:
  - separate inequalities depending on polarity of x:
    - $A_i \le \alpha_i x$
    - $\beta_j x \le B_j$
  - define values $a = \max_i \lceil A_i/\alpha_i \rceil$ and $b = \min_j \lceil B_j/\beta_j \rceil$
- if b is defined, return x = b else return x = a
- further continue with the conjunction of all formulas $\lceil A_i/\alpha_i \rceil \le \lceil B_j/\beta_j \rceil$

$(x, y) \Rightarrow$ c.

$2y - b \le 3\mathbf{x} + a \wedge 2\mathbf{x} - a \le 4y + b$ example

$2y - b - a \le 3x \quad \wedge \quad 2x \le 4y + a + b$

$4y - 2b - 2a \le 6x \le 12y + 3a + 3b$

$(4y - 2b - 2a) / 6 \le x \le (12y + 3a + 3b) / 6$

$(4y - 2b - 2a) / 6 \le \lfloor (12y + 3a + 3b) / 6 \rfloor$

two extra variables:

$(4y - 2b - 2a) / 6 \le l \wedge 12y + 3a + 3b = 6 * l + k$

$\wedge\ 0 \le k \le 5$

$4y - 2b - 2a \le 6 * l \wedge \mathbf{12y + 3a + 3b = 6 * l + k}$

$\wedge\ 0 \le k \le 5$

$6l = 12y + 3a + 3b - k$

pre: $6 \mid 3a + 3b - k + 12y$

$4y - 2b - 2a \le 12y + 3a + 3b - k$

i.e.

$-5b - 5a + k \le 8y \quad \rightarrow y = \lceil (k - 5a - 5b)/8 \rceil$

Generated Code Contains Loops

```
val (x1, y1) = choose(x: Int, y: Int =>
                    2*y − b =< 3*x + a &&  2*x − a =< 4*y + b)
```

```
val kFound = false
 for k = 0 to 5 do {
   val v1 = 3 * a + 3 * b − k
   if (v1 mod 6 == 0) {
     val alpha = ((k − 5 * a − 5 * b)/8).ceiling
     val l = (v1 / 6) + 2 * alpha
     val y = alpha
     val kFound = true
     break } }
 if (kFound)
   val x = ((4 * y + a + b)/2).floor
 else throw new Exception(”No solution exists”)
```

NP-Hard Constructs

- Divisibility combined with inequalities:
  - corresponding to big disjunction in q.e. , we will generate a for loop with constant bounds (could be expanded if we wish)
- Disjunctions
  - Synthesis of a formula computes program and exact precondition of when output exists
  - Given disjunctive normal form, use preconditions to generate if-then-else expressions (try one by one)

Synthesis for Disjunctions

$$
\begin{array}{l}
[\![\vec{x}, D_1 \vee \ldots \vee D_n]\!] = \\
\quad \texttt{let}\ (\mathsf{pre}_1, \vec{\Psi}_1) = [\![\vec{x}, D_1]\!] \\
\qquad\qquad \ldots \\
\qquad\qquad (\mathsf{pre}_n, \vec{\Psi}_n) = [\![\vec{x}, D_n]\!] \\
\quad \texttt{in} \\
\left(\bigvee_{i=1}^{n} \mathsf{pre}_i, \left\{ \begin{array}{l}
\qquad \mathsf{if}\ (\mathsf{pre}_1)\ \vec{\Psi}_1 \\
\mathsf{else\ if}\ (\mathsf{pre}_2)\ \vec{\Psi}_2 \\
\ldots \\
\mathsf{else\ if}\ (\mathsf{pre}_n)\ \vec{\Psi}_n \\
\mathsf{else\ assert(false)}
\end{array} \right\} \right)
\end{array}
$$

General Form of Synthesized Functions for Presburger Arithmetic

choose x **such that** F(x,a) $\rightarrow$ x = t(a)

Result t(a) is expressed in terms of

+, C*, /C, %C, **if**

Need arithmetic for solving equations

Need conditionals for

- disjunctions in input formula
- divisibility and inequalities (find a witness meeting bounds and divisibility by constants)

t(a) = if P_1(a) t_1(a) elseif … elseif P_n(a) t_n(a)
else error("No solution exists for input",a)

Methodology QE $\rightarrow$ Synthesis

- Each quantifier elimination 'trick' we found corresponds to a synthesis trick
- Find the corresponding terms
- Key techniques:
  - one point rule immediately gives a term
  - change variables, using a computable function
  - strengthen formula while preserving realizability
  - recursively eliminate variables one-by one
- Example use
  - transform formula into disjunction
  - strengthen each disjunct using equality
  - apply one-point rule

Compile-time warnings

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  choose((h: Int, m: Int, s: Int) ⇒ (
        h * 3600 + m * 60 + s == totalSeconds
    && h ≥ 0 && h < 24
    && m ≥ 0 && m < 60
    && s ≥ 0 && s < 60
  ))
```

```
Warning: Synthesis predicate is not
satisfiable for variable assignment:
  totalSeconds = 86400
```

Compile-time warnings

```
def secondsToTime(totalSeconds: Int) : (Int, Int, Int) =
  choose((h: Int, m: Int, s: Int) ⇒ (
        h * 3600 + m * 60 + s == totalSeconds
    && h ≥ 0
    && m ≥ 0 && m < 60
    && s ≥ 0 && s ≤ 60
  ))
```

```
Warning: Synthesis predicate has multiple
solutions for variable assignment:
  totalSeconds = 60
Solution 1: h = 0, m = 0, s = 60
Solution 2: h = 0, m = 1, s = 0
```