Viktor Kuncak, Mikaël Mayer, Ruzica Piskac, Philippe Suter
Laboratory for Automated Reasoning and Analysis

Comfusy Complete Functional Synthesis

Correct-by-construction code synthesis for Scala

```
val totalSecs: Int = Console.readInt

val (hours,minutes,seconds) = choose((h,m,s) => (
     0 ≤ m && m < 60
  && 0 ≤ s && s < 60
  && 3600 * h + 60 * m + s == totalSecs
))
```

```
val (hours, minutes, seconds) = {
  val t1 = totalSecs / 3600
  val t2 = totalSecs + (-3600 * t1)
  val t3 = Math.min(t2 / 60, 59)
  val t4 = totalSecs + (-3600 * t1) + (-60 * t3)
  (t1,t3,t4)
}
```

Users write **implicit** definitions

Comfusy generates **explicit** code

The definitions combine **known** variables with variables whose values will be **synthesized**.

Comfusy computes a **function** in terms of the known variables that expresses the new values.

Synthesis: Comfusy starts from an implicit specification, and involves compile-time precomputation.

Functional: it computes a function that satisfies a given input/output relation.

Complete: it is guaranteed to work for all specification expressions from a well-specified class.

Comfusy is implemented as a compiler plugin for the Scala programming language.

The plugin and its documentation are freely available at:

`http://lara.epfl.ch/dokuwiki/comfusy`

Further examples

```
val (a1,a2) = choose((a1: Set[0], a2: Set[0]) =>
     a1 union a2 == s
  && a1 intersect a2 == Set.empty
  && a1.size - a2.size ≤ 1
  && a2.size - a1.size ≤ 1)
```

Synthesis for **set expressions**

```
val ratio = choose((r: Int) =>
     a == r * b
  || b == r * a)
```

Non-linear arithmetic

```
def pow(base: Int, p: Int) = {
  def fp(m: Int, b: Int, i: Int) = i match {
    case 0 => m
    case 2*j => fp(m, b*b, j)
    case 2*j+1 => fp(m*b, b*b, j)
  }
  fp(1, base, p)
}
```

Expressive **pattern matching**

A call of the form

$$\vec{r} = \texttt{choose}(\vec{x} \Rightarrow F(\vec{a},\vec{x}))$$

corresponds to solving the problem

$$\forall \vec{a}.\exists \vec{x}.F(\vec{a},\vec{x})$$

Our approach differs from calling a solver at runtime because part of the computation is done at **compile time**.

The algorithmic basis is **quantifier elimination**.

References
V. Kuncak, M. Mayer, R. Piskac, P. Suter, Complete Functional Synthesis, PLDI 2010
V. Kuncak, M. Mayer, R. Piskac, P. Suter, Comfusy: A Tool for Complete Functional Synthesis (Tool Presentation), CAV 2010