ArchiType – Component & Connector Synthesis

B. Düdder: Automatic Synthesis of Component & Connector-Software Architectures with Bounded Combinatory Logic
Robot Control Programs

Bessai, Dudenhefner, Düdder, Martens, J.R.:
Combinatory Process Synthesis, ISOLA 2016 [Bes+16]
Combinatory Process Synthesis

Fig. 2: Taxonomies for robot shapes and robot movement

<table>
<thead>
<tr>
<th>Constants</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>car, caterpillar, humanoid, insectoid</td>
<td>Robot shapes</td>
</tr>
<tr>
<td>followsLine, followsWall</td>
<td>Robot jobs</td>
</tr>
<tr>
<td>oneLightSensor, twoLightSensors, twoUltrasoundSensors</td>
<td>Sensor configuration</td>
</tr>
<tr>
<td>stopsOnTouch, stopsOnLight, stopsOnSound</td>
<td>Stop conditions</td>
</tr>
</tbody>
</table>

Table 2: Semantic constants for different variability dimensions

<table>
<thead>
<tr>
<th>Constants</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set, read</td>
<td>Sensor functions</td>
</tr>
<tr>
<td>abort</td>
<td>Job abort condition</td>
</tr>
<tr>
<td>turnLeft, turnRight, moveForward, stop</td>
<td>Movement types</td>
</tr>
</tbody>
</table>

Table 3: Semantic constants in the domain of LEGO NXT robots

1. setSensors': □(subproc ∩ twoLightSensors ∩ stopsOnTouch ∩ set) a subprocess to set the light and touch sensors.
2. executeJob': □(subproc ∩ twoLightSensors ∩ stopsOnTouch ∩ car ∩ followsLine) ∩ jobProc a subprocess that makes the robot execute the job until aborted.
3. stop': □(subproc ∩ car ∩ stop) a subprocess that makes a car robot stop.

createRobotProgram(setSensors', executeJob', stop'): robotProgram
∩ □(proc ∩ twoLightSensors ∩ stopsOnTouch ∩ car ∩ followsLine)
**Inhabitation** Given the repository $\mathcal{D}$ of typed PCs and typed CCs, asking the inhabitation question $\mathcal{D} \vdash ?: \Box(\text{proc} \land \text{car} \land \text{followsLine} \land \text{twoLightSensors} \land \text{stopsOnTouch}) \land \text{robotProgram}$ automatically synthesizes the following applicative term:

```plaintext
createRobotProgram(
    setSensors(box setTwoLightsSensors, taskToSubProc(box setTouchSensor)),
    executeJob(taskToSubProc(box readTouchSensor),
                box abortConditionTouchSensor, box readTwoLightSensors,
                conditionalMove(box turnLeftConditionLineFollowerTwoLightSensors,
                                 box turnRightConditionLineFollowerTwoLightSensors),
                box turnLeftCar, box turnRightCar, box moveForwardCar),
    box stopCar)
```

*Fig. 4: Inhabited BPMN 2.0 process*
LaunchPad – Feature-Oriented Program Synthesis by George T. Heineman using CLS

Heineman, Bessai, Düdder, Martens, J.R.: A Long and Winding Road Towards Modular Synthesis, ISOLA 2016 [Hei+16]
Γ = {
  customerForm : (String → java.net.URL → OptionSelection → Form)
  dropDownSelector : (java.net.URL → OptionSelection)
  radioButtonSelector : (java.net.URL → OptionSelection)
  companyTitle : String
  databaseLocation : java.net.URL
  logoLocation : java.net.URL
  alternateLogoLocation : java.net.URL
}
\[ \Gamma = \{ \text{customerForm} : (\text{String} \rightarrow \text{java.net.URL} \rightarrow \text{OptionSelection} \rightarrow \text{Form}) \cap \\
\quad (\text{Title} \rightarrow \text{Location(Logo)} \rightarrow \text{ChoiceDialog}(\alpha) \rightarrow \text{OrderMenu}(\alpha)) \}
\]

\[ \text{dropDownSelector} : (\text{java.net.URL} \rightarrow \text{OptionSelection}) \cap \\
\quad (\text{Location(Database)} \rightarrow \text{ChoiceDialog(DropDown)}) \]

\[ \text{radioButtonSelector} : (\text{java.net.URL} \rightarrow \text{OptionSelection}) \cap \\
\quad (\text{Location(Database)} \rightarrow \text{ChoiceDialog(RadioButtons)}) \]

\[ \text{companyTitle} : \text{String} \cap \text{Title} \]

\[ \text{databaseLocation} : \text{java.net.URL} \cap \text{Location(Database)} \]

\[ \text{logoLocation} : \text{java.net.URL} \cap \text{Location(Logo)} \]

\[ \text{alternateLogoLocation} : \text{java.net.URL} \cap \text{Location(Logo)} \} \]
Combinator Interface in Scala

trait Repository {
  type Form
  type OptionSelection

  lazy val alpha = Variable("alpha")
  lazy val `kinding` = Kinding(alpha).addOption('DropDown').addOption('RadioButton')

trait CustomerForm {
  def apply(title: String, logoLocation: URL, optionSelector: OptionSelection): Form
  val semanticType: Type = `Title` => `Location` => `ChoiceDialog`(alpha) => `OrderMenu`(alpha)
}

val customerForm: CustomerForm
class SwingRepository extends Repository {
  type Form = CompilationUnit
  type OptionSelection = CompilationUnit => CompilationUnit

@combinator object customerForm extends CustomerForm {
  override def apply(title: String, logoLocation: URL, optionSelector: OptionSelection): CompilationUnit = {
    val file = scala.io.Source.fromInputStream(getClass.getResourceAsStream("CustomerForm.java")).mkString
    val form = Java(form).compilationUnit()
    val cls = form.getClassByName("CustomerForm").get
    val initMethod = cls.getMethodByName("initComponents").get(0)

    optionSelector(form)

    form.addImport("java.net.URL")
    initMethod
      .getBody.get()
      .addStatement(0, Java{
        s""
        try {
          this.add(new JLabel(new ImageIcon(new URL("$logoLocation"))));
        } catch (Exception e) {
          
          s"""\n        }""".stripMargin).statement()

    initMethod
      .getBody.get()
      .addStatement(0, Java(s"""this.setTitle("$title");""" ).statement())

    form
  }
}
Variations
Web Interface to Inhabitation Service

**Requests:**

```java
```

**Solutions:**

**Variation 0:**

```java
List[Tree(customerForm,List(Tree(companyTitle,List())), Tree(logoLocation,List()), Tree(dropDownSelector,List(Tree(databaseLocation,List()))))]
```

**Variation 1:**

```bash
# clone into new git:
git clone -b variation_1 http://localhost:9000/guidemo/guidemo.git
# checkout branch in existing git:
git fetch origin
git checkout -b variation_1 origin/variation_1
```

**Variation 2:**

Compute

**Variation 3:**

Compute
CLS Demo

See the movie ...