Formalising Simple Codecharts

**Introduction**

Codecharts [1] are a diagrammatic notation for describing the syntactic relationship between classes in a codebase. Our work formalises the syntax of a subset of codecharts.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>A <strong>rectangle</strong> represents a class, labelled by a class name.</td>
</tr>
<tr>
<td>Set of</td>
<td>An <strong>offset rectangle</strong> represents a set of classes, labelled by a set of class names.</td>
</tr>
<tr>
<td>Classnames</td>
<td>An <strong>ellipse</strong> represents a method signature, labelled by a signature name.</td>
</tr>
<tr>
<td>Set of</td>
<td>An <strong>offset ellipse</strong> represents a set of method signatures, labelled by a set of signature names.</td>
</tr>
<tr>
<td>Signatures</td>
<td>A <strong>triangle</strong> represents an inheritance class hierarchy, labelled by a class hierarchy name.</td>
</tr>
<tr>
<td>Hiearchies</td>
<td>An <strong>offset triangle</strong> represents a set of inheritance class hierarchies, labelled by a set of class name hierarchies.</td>
</tr>
<tr>
<td>Inversion</td>
<td>An <strong>inverted triangle</strong> represents an unary relation, labelled by a unary relation name.</td>
</tr>
<tr>
<td>Class</td>
<td>A <strong>single-headed arrow</strong> represents a relationship between the source and target, labelled by a binary relation name.</td>
</tr>
<tr>
<td>Target</td>
<td>A <strong>double-headed arrow</strong> represents a pairwise relationship between the source and target, labelled by a binary relation name.</td>
</tr>
</tbody>
</table>

Our work allows us to determine whether a codechart is syntactically well-formed or non well-formed.

**An Abstract Syntax**

We proposed an abstract syntax of codecharts:

\[
(R, T_s, E, OR, OT_s, OE, M, T_a, A_r)\]

where \(R\) is the set of all class names used to label rectangles, \(OR\) is a set of sets of class names that label offset rectangles, \(T_s\) is the set of class name hierarchies used to label triangles, \(T_a\) captures unary relations and \(A_r\) captures binary relations. The other components are described in the paper.

The above example contains a rectangle, an offset rectangle, a single headed arrow and an inverted triangle. Its abstract syntax can be captured as follows:

\[
\begin{align*}
    R &= \{ \text{WhiteRhino, BlackRhino, Animal} \} \\
    OR &= \{ \{ \text{WhiteRhino, BlackRhino} \} \} \\
    T_s &= \{ (\text{Endangered}, \{ \text{WhiteRhino, BlackRhino} \}) \} \\
    T_a &= \{ \{ \text{WhiteRhino, BlackRhino}, is\, a, \text{Animal} \} \}
\end{align*}
\]

The above example contains a triangle. Its abstract syntax can be captured as follows:

\[
\begin{align*}
    R &= \{ \text{WhiteRhino, BlackRhino, Animal} \} \\
    T_s &= \{ (\text{Animal}, \{ \text{WhiteRhino, BlackRhino} \}) \}
\end{align*}
\]

**Well-Formed Codecharts**

Future work includes a new concrete syntax that facilitates formal reasoning over how a codechart is drawn.

The above codechart contains three labelled rectangles representing classes, two labelled single headed arrows specifying that Music and Film both \textit{Inherit} from Media, and a labelled inverted triangle specifying that Media is \textit{Abstract}. The concrete syntax representation of this is given on the right, where \(R\) is a set of rectangles, \(T_s\) is a set of inverted triangles, \(A_r\), is a set of single headed arrows, and all other sets are empty.

**Non Well-Formed Codecharts**

The proposed concrete syntax will allow us to decide when a codechart is not well-formed, such as below. The first is not well-formed because \textit{getStock} does not overlap a rectangle, the second because the \textit{Inherit} arrow has no target, and the third because \textit{Abstract} overlaps two shapes.

**References**