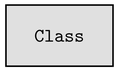
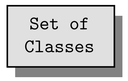
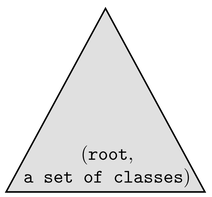
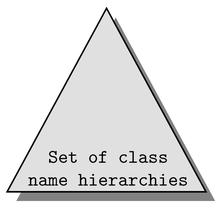
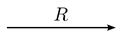
Formalising Simple Codecharts

Jon Nicholson and Aidan Delaney
Visual Modelling Group, University of Brighton

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.

Syntax	Represents
	A **rectangle** represents a class, labelled by a class name.
	An **offset rectangle** represents a set of classes, labelled by a set of class names.
	An **ellipse** represents a method signature, labelled by a signature name.
	An **offset ellipse** represents a set of method signatures, labelled by a set of signature names.
	A **triangle** represents an inheritance class hierarchy, labelled by a class hierarchy name.
	An **offset triangle** represents a set of inheritance class hierarchies, labelled by a set of class name hierarchies.
	An **inverted triangle** represents an unary relation, labelled by a unary relation name.
	A **single-headed arrow** represents a relationship between the source and target, labelled by a binary relation name.
	A **double-headed arrow** represents a pairwise relationship between the source and target, labelled by a binary relation name.

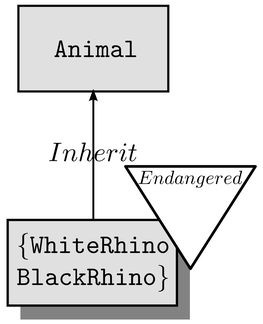
Our work allows us to determine whether a codechart is syntactically well-formed or non well-formed.

An Abstract Syntax

We proposed an abstract syntax of codecharts:

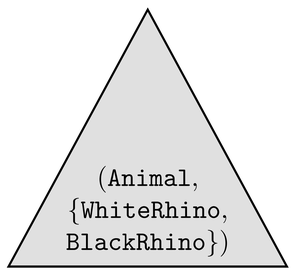
$$(\mathcal{R}, \mathcal{T}_{\triangle}, \mathcal{E}, \mathcal{OR}, \mathcal{OT}_{\triangle}, \mathcal{OE}, \mathcal{M}, \mathcal{T}_{\triangledown}, \mathcal{A}_{\blacktriangleright}, \mathcal{A}_{\blacktriangleright\blacktriangleright})$$

where $\mathcal{R}$ is the set of all class names used to label rectangles, $\mathcal{OR}$ is a set of sets of class names that label offset rectangles, $\mathcal{T}_{\triangle}$ is the set of class name hierarchies used to label triangles, $\mathcal{T}_{\triangledown}$ captures unary relations and $\mathcal{A}_{\blacktriangleright}$ 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:

- $\mathcal{R}$ = {WhiteRhino, BlackRhino, Animal}
- $\mathcal{OR}$ = {{WhiteRhino, BlackRhino}}
- $\mathcal{T}_{\triangledown}$ = {(*Endangered*, {WhiteRhino, BlackRhino})}
- $\mathcal{A}_{\blacktriangleright}$ = {({WhiteRhino, BlackRhino}, *is a*, Animal)}

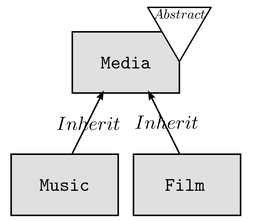


The above example contains a triangle. Its abstract syntax can be captured as follows:

- $\mathcal{R}$ = {WhiteRhino, BlackRhino, Animal}
- $\mathcal{T}_{\triangle}$ = {(Animal, {WhiteRhino, BlackRhino})}

Well-Formed Codecharts

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

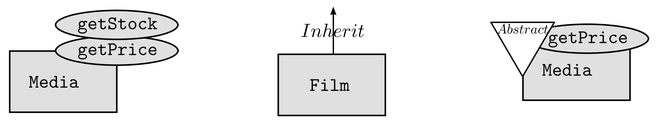


- $\boldsymbol{R}$ = $\{r_1, r_2, r_3\}$, $\boldsymbol{T}_{\triangledown}$ = $\{t_1\}$, $\boldsymbol{A}_{\blacktriangleright}$ = $\{a_1, a_2\}$,
- $\boldsymbol{\lambda}$, a labelling function where $\boldsymbol{\lambda}(r_1)$ = Media, $\boldsymbol{\lambda}(r_2)$ = Music, $\boldsymbol{\lambda}(r_3)$ = Film, $\boldsymbol{\lambda}(t_1)$ = ***Abstract***, $\boldsymbol{\lambda}(a_1)$ = ***Inherit***, and $\boldsymbol{\lambda}(a_2)$ = ***Inherit***.

The above codechart contains three labelled rectangles representing classes, two labelled single headed arrows specifying that Music and Film both ***Inherit*** from Media, and a labelled inverted triangle specifying that Media is ***Abstract***. The concrete syntax representation of this is given on the right, where $\boldsymbol{R}$ is a set of rectangles, $\boldsymbol{T}_{\triangledown}$ is a set of inverted triangles, $\boldsymbol{A}_{\blacktriangleright}$ 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 getStock does not overlap a rectangle, the second because the ***Inherit*** arrow has no target, and the third because ***Abstract*** overlaps two shapes.



References

[1] A.H. Eden and J. Nicholson.
Codecharts: Roadmaps and Blueprints for Object-Oriented Programs.
Wiley-Blackwell, April 2011.