Modeling and Reasoning with Multirelations, and their encoding in Alloy

Peiyuan Sun, Zinovy Diskin, Michał Antkiewicz, and Krzysztof Czarnecki
University of Waterloo

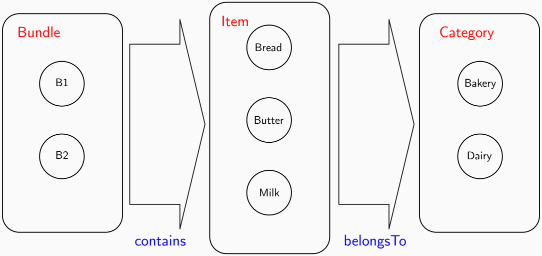
Relations

- **Relation** is a fundamental concept in modeling languges
    - Association in UML and OCL
    - Unary/Binary/Ternary relation in Alloy
- **Relation** is usually refered to ordinary relation, where objects can be related no more than once.
- **Multirelation** naturally arises in domain modeling, where objects can be related multiple times.

Motivation

A seasonal sale scenario

The manager of a grocery store asks employee to prepare some bundles for the coming seasonal sale. A bundle contains several food items, and each item belongs to a certain product category.



Rules on bundling

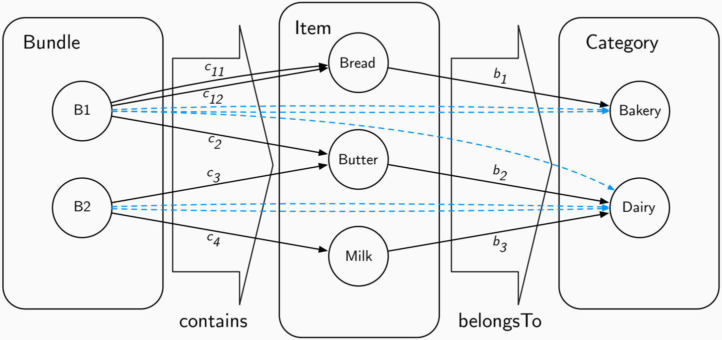
The manager also imposes some rules on the bundle content:

1. Every bundle must contain at least two dairy products.
2. Every bundle must contain items from at least two product categories.

Rules can be applied in two ways:

- Validation: check if an existing bundle setting is valid.
- Synthesis: generate valid bundle settings from the rules.

A bundling instance



- The objects in Bundle and Item should be able to be related more than once, which forms a multirelation.
- In order to observe how many dairy products in each bundle, composing relations *contains* and *belongsTo* should result in a multirelation (perserving multiplicity).

Work with multiconcepts

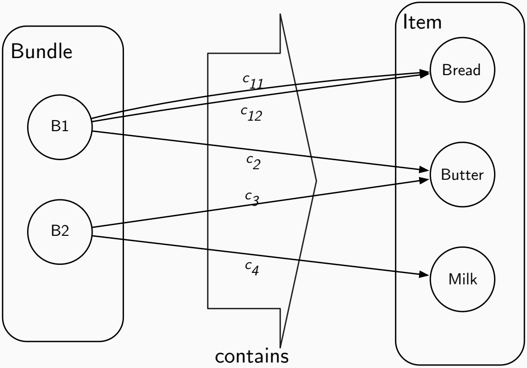
To work with multiconcepts(multiset/bag and multirelation), we need:

- Directly declare a multirelation.
- Operations over multiconcepts such as composition.
- Can control the result of an operation to be multi or not.

Most modeling languages do not have a first-class support on multiconcepts, so there is no direct way to work with multiconcepts and often encoding is needed. We would like to develop a general solution for encoding multiconcepts.

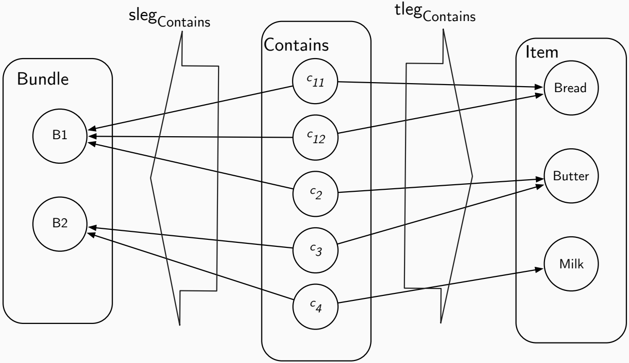
Formalization

Before reification



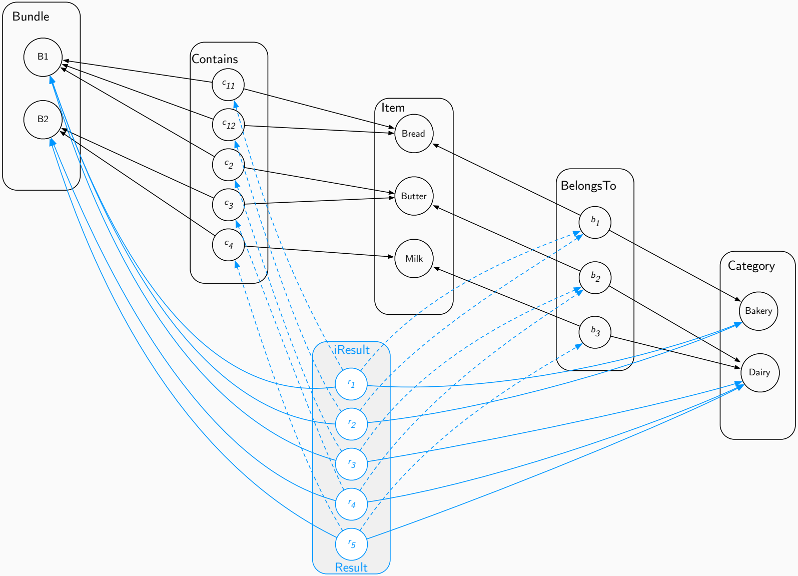
The basic idea of the formalization is to reify links as objects.

After reification

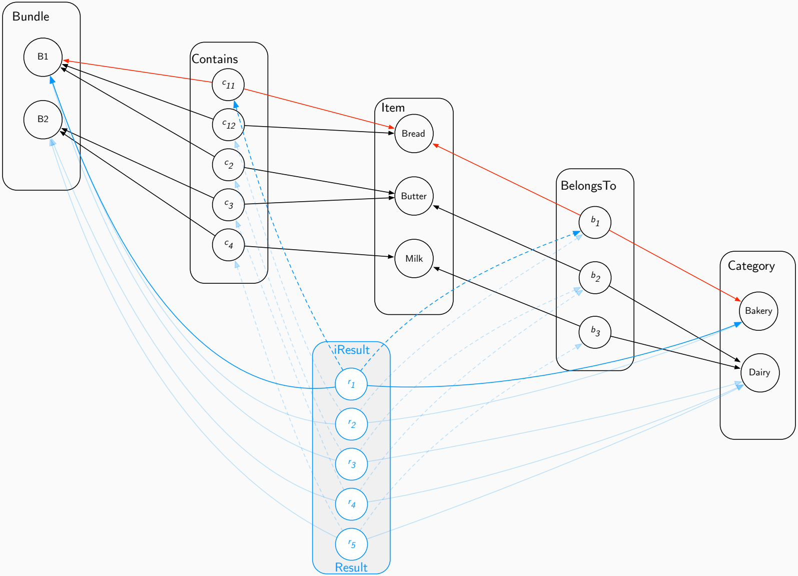


By reification, an index set *Contains* is introduced, in which the elements represent links, along with two total functions (source leg and target leg) pointing to the domain and range of the original multirelation. The whole shape is called a *span*.

Full bundling model after reification



Composition



Formalization

A mathematical framework based on category theory, including concepts:

- Family: t_1
- Span: $(Contains, s_1, t_1)$
- Pullback: $(Result, p_1, p_2)$
- Family/Span Composition

$$\begin{tikzcd}
Bundle & & \\
Contains \arrow[u, "s_1"] \arrow[r, "t_1"] & Item & \\
Result \arrow[u, "p_1"] \arrow[r, "p_2"'] \arrow[uu, bend left, "s"] \arrow[rr, bend right, "t"'] \arrow[ur, phantom, "\urcorner", very near start] & BelongsTo \arrow[u, "s_2"'] \arrow[r, "t_2"'] & Category
\end{tikzcd}$$

Diagram of span composition

A Multiconcept Library in Alloy

Features

- Use multiconcepts in demand without changing existing model.
- Declarative style to use multiset and multirelation with parametric module.
- Operations over multiconcepts:
    - composition
    - multiplicity/cardinality
    - max-union, min-intersection, merge
    - domain restriction, range restriction, inverse
    - lift, drop
    - traditional transitive closure is not implemented since it leads to infinity multiplicity in certain cases.
- Theme settings which provides human-readable visualization of instances.

Declare a multirelation

Example

```
open mrel[Bundle, Item] as Contain
open mrel[Item, Category] as BelongsTo
open mrel[Bundle, Catgory] as Result
```

- Open the module `mrel` with source and target type parameters to declare a multirelation, assign a name for future reference.

Compose multirelations

Example

```
fact {
  BelongsTo/liftedFrom[belongsTo]
  Result/composedFrom[Contain/get, BelongsTo/get]
}
```

- We could lift a ordinary binary relation to a multirelation representation.
- To perform a composition, a new `mrel` need to be declared to hold the result.

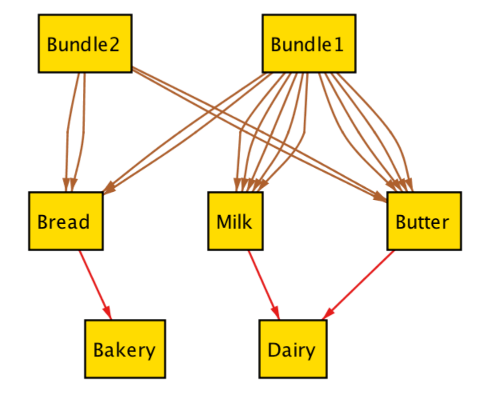
Specify bundling rules

Example

```
fact {
  all b : Bundle | #(b <: Result/get :> Dairy) >= 2
  all b : Bundle | #(b <: (drop[Result/get]))  >= 2
}
```

- More importantly, We can express the bundling rules in the model.

Visualize in Alloy Analyzer



Conclusion

Contribution

- A category-theory based multiconcepts framework as a theoretical contribution to the area of MDE.
- An Alloy multiconcept library which enable the Alloy user to easily integrate multiconcepts into the model as a contribution to Alloy community.
- Available in Github OCL repository: https://github.com/jcabot/ocl-repository

Future work

- Numeric-based multiconcepts encoding in Alloy.
- A multiconcepts implementation for SMT Solver where sets and total functions are available.
- Modeling language with first-class multiconcepts support, such as Clafer.

Questions?