Oruga: An Avatar of Representational Systems Theory

Daniel Raggi[1,*], Gem Stapleton[1], Mateja Jamnik[1], Aaron Stockdill[2], Grecia Garcia Garcia[2] and Peter C.-H. Cheng[2]

[1]*University of Cambridge, Cambridge, UK*
[2]*University of Sussex, Brighton, UK*

Abstract
Humans use representations flexibly. We draw diagrams, change representations and exploit creative analogies across different domains. We want to harness this kind of power and endow machines with it to make them more compatible with human use. Previously we developed Representational Systems Theory (RST) to study the structure and transformations of representations [1, 2]. In this paper we present ORUGA (*caterpillar* in Spanish; a symbol of transformation), an implementation of various aspects of RST. ORUGA consists of a core of data structures corresponding to concepts in RST, a language for communicating with the core, and an engine for producing transformations using a method we call *structure transfer*. In this paper we present an overview of the core and language of ORUGA, with a brief example of the kind of transformation that structure transfer can execute.

Keywords
Representation, Transformation, Heterogeneous reasoning

1. Introduction: rep2rep and RST

This work is part of the rep2rep project [3, 4], whose aim is to study and implement systems that mimic and accommodate the flexibility of human representational skills. As part of rep2rep, we developed Representational Systems Theory (RST) [1, 2], a theoretical foundation for studying the structure and transformations of representations. The main requirement for such a theory is that it must be general enough to account for the diversity of representations used by humans (e.g., formal and natural languages, geometric figures, graphs, plots, etc.), and at the same time it must be rigorous and precise enough to be implementable. For example, it must be able to explain the relation between the validity of $1 + 2 = 3$ and the fact that $\overset{\circ}{\circ\circ}$ can be built by joining $\circ\circ$ and $\circ$, and ultimately allow us to produce such transformations between arithmetic terms and dot diagrams; but its scope must not be limited to arithmetic and dot diagrams.

One of the key innovations of RST is the notion of a *construction space*, where many concepts of interest for the study of representations can be defined in graph-theoretic terms. Notably, the concept of a *construction* generalises that of a syntax tree, but its weaker restrictions and

HLC 2022: 3rd International Workshop on Human-Like Computing, September 28–30, 2022, Cumberland Lodge, UK
[*]Corresponding author.
daniel.raggi@cl.cam.ac.uk (D. Raggi)
0000-0002-9207-6621 (D. Raggi); 0000-0002-6567-6752 (G. Stapleton); 0000-0003-2772-2532 (M. Jamnik); 0000-0003-3312-5267 (A. Stockdill); 0000-0002-7327-7225 (G. Garcia Garcia); 0000-0002-0355-5955 (P. C.-H. Cheng)

our graph-theoretic approach allow us to model more complex structures and inspect their properties. Importantly, it allows us to model representations often considered informal, and to do so uniformly across different representational systems so that we can encode relations and produce transformations between them.

In this paper we present an overview of Oruga's core data structures and language for communicating with the core. Specifically, we demonstrate how some of the main concepts of RST are declared in Oruga. Here we do not focus on the engine for producing transformations. The Standard ML code can be found in [5].

2. The core of Oruga

Oruga's core data structures are *type systems*, *constructor specifications*, *constructions* and *transfer schemas*. These are crucial for specifying construction spaces, building structures within them, and producing transformations across them.

2.1. Type Systems

In RST we refer to concrete representations as *tokens*, and we assign them *types*. This induces equivalence classes of tokens apropos to the token-type dichotomy [6]. RST is agnostic concerning the criteria for determining whether two tokens have the same type – it simply regards *type* as a function that assigns a value to every token. For example, we may say that the arithmetic expression $1 + 1$ contains two tokens of type `one`. RST also enables subtyping via a partial order on the set of types. For instance, we can set the order to be such that `one` is a subtype of `numeral`, and `numeral` is a subtype of `numExp`. Formally, we define a *type system* as a pair, $(Ty, \leq)$, where Ty is a set whose elements are called types, and $\leq$ is a partial order over Ty. In the Oruga language, we can declare type systems, as demonstrated here (right): a type system for a fragment of arithmetic. Expressions such as `_:var` declare that the type `var` has infinitely many subtypes which are not explicitly declared. In practice, it means that the user of Oruga can write `t:A:var`, and this means `t` is a token of type `A`, which is a subtype of `var`. The transitive/reflexive closure of the subtype relation is calculated in the background to facilitate minimal declarations.

```
typeSystem arithT =
  types _:numeral, _:var, _:numExp, _:formula,
        plus, minus, binOp, leq, equals, binRel
  order var < numExp, numeral < numExp,
        plus < binOp, minus < binOp,
        leq < binRel, equals < binRel
```

2.2. Constructor specifications

A construction space is where we encode how tokens are constructed, for example, how $1 + 2$ relates to 1, $+$ and 2. Formally, a construction space is a triple (T, C, G) where T is a type system, C is a *constructor specification*, and G is a *structure graph*. We explain these below.

Formally, a constructor specification is a pair (Co, sig) where Co is a set of elements called *constructors* and sig is a function with domain Co that, given a constructor, returns a pair $([\tau_1, \ldots, \tau_n], \tau)$, where $[\tau_1, \ldots, \tau_n]$ is a finite sequence of *input* types and τ is an *output* type.

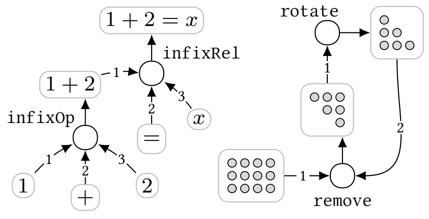
For example, a constructor that infixes a binary operator, `infixOp`, may be defined so that $sig(\texttt{infixOp}) = ([\texttt{numExp}, \texttt{binOp}, \texttt{numExp}], \texttt{numExp})$.

See (right) how we declare a finite constructor specification `arith` for type system `arithT` in the Oruga language.

```
conSpec arith:arithT =
  infixOp : [numExp,binOp,numExp] -> numExp,
  infixRel : [numExp,binRel,numExp] -> formula,
  implicitMult : [numExp,numExp] -> numExp
```

The structure graph associated with a construction space is the home of *all* admissible constructions of *every* token of the construction space. This is where the structure of representations is encoded. See [1] for a full formal definition. For most interesting construction spaces, its graph is infinite and perhaps undecidable (i.e., we cannot know if any arbitrary graph is a part of it). Then, insofar as it concerns implementation we need to think about manageable parts of structure graphs. This leads us to the concept of a *construction*.

2.3. Constructions and patterns

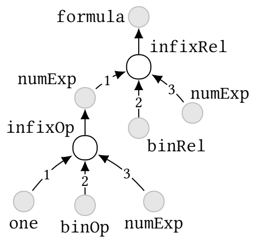


A construction captures *one* way in which *one* token is constructed. Here (right) we show two constructions, one for $1 + 2 = x$ and another for dot diagram .

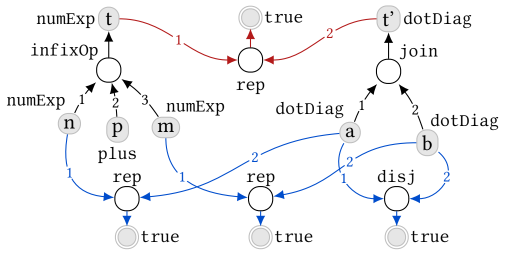
Constructions have many useful properties; in particular, they can be easily encoded with a recursive datatype (for implementation, see [5]). See (right) how a construction is declared. Notation `t1:1plus2:numExp` creates a new type, `1plus2`, such that `1plus2` is a subtype of `numExp`; this is allowed because we declared `_:numExp` in the type system.

```
construction con:arith =
  t:1plus2equalsx:formula
    <- infixRel[t1:1plus2:numExp
                  <- infixOp[t11:1:numeral,
                             t12:plus,
                             t13:2:numeral],
                t2:equals,
                t3:x:var]
```

A *pattern* for a construction space, (T, C, G), is a construction that satisfies the restrictions given by the type system and constructor specification, but may not necessarily be a part of G. Patterns are useful given the concept of *matching*, as they allow us to capture classes of constructions. Roughly, we say that a construction matches a pattern if there exists an isomorphism from the former to the latter that respects the subtype order. See (right) an example of a pattern; the labels on token vertices specify types. The construction of $1 + 2 = x$ (above) matches this pattern.



2.4. Transfer schemas



One key concept for achieving transformations is that of a *transfer schema*. A transfer schema is, roughly, an inference rule for deriving relations that cross construction spaces. We omit the formal definition here. See (right) a transfer schema which captures the fact that, provided that two disjoint dot diagrams (`a` and `b`) represent two numerical expressions (`n` and `m`), then the result of joining them yields a representation of `n` + `m`.

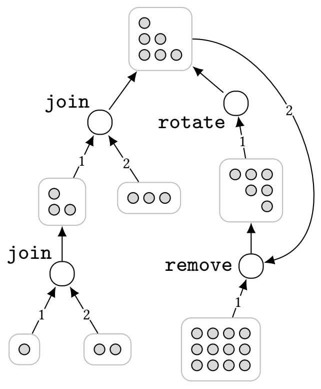
A transfer schema is declared in ORUGA by specifying source and target patterns, and the antecedent and consequent constraints, as demonstrated here (right).

```
tSchema plusJoin:(arith,dotDiagrams) =
  source t:numExp <- infixOp[n:numExp,
                             p:plus,
                             m:numExp]
  target t':arr <- join[a:arr,b:arr]
  antecedent ([n:numExp],[a:arr]) :: rep,
             ([m:numExp],[b:arr]) :: rep,
             ([],[a:arr,b:arr]) :: disj
  consequent ([t:numExp],[t':arr]) :: rep
```

3. Structure Transfer

Structure transfer is a method for producing transformations of a given graph in a source construction space into some target construction space. The goal of structure transfer is to satisfy some constraint involving a given token and some sought-after token. For instance, if we start with token $1 + 2 + 3$ and we wish to find a dot arrangement which *represents* it, structure transfer will use transfer schemas to try to build such dot arrangement while simultaneously proving that the desired constraint must hold. A general version of this method is presented in [2].

We have had success with various tests of structure transfer. For example, it is possible to define a transfer schema that roughly specifies that a dot arrangement represents an equation if the same arrangement represents each side of the equation. Thus, given $1 + 2 + 3 = 3(3 + 1)/2$, structure transfer will try to find one arrangement that can be constructed in two ways, one corresponding to $1 + 2 + 3$ and the other to $3(3 + 1)/2$. One result is a pair of constructions of arrangement , as shown here (right). The generalisation of this result is a graphical proof of Gauss' sum.



4. Past and future work

ORUGA's purpose is to facilitate encoding diverse representations within a uniform framework so that we can perform transformations between them. So far, we have implemented a restricted version of the methods presented in [2], wherein transfer schemas are used as inference rules applied backwards (from the goal). Our approach is domain-independent. To date we have used ORUGA to transform across multiple construction spaces (e.g., arithmetic, Euler diagrams, set algebra, propositional logic and geometry). We discuss its generality in [2], and in particular its relation to similar but more specific formal methods [7, 8, 9], as well as its relation to the application and discovery of analogies [10]. In future work we aim to explore more in depth the potential of structure transfer for analogy, and other applications of RST in cognitive science. We are currently developing a graphical interface to improve its usability, especially for inputting constructions.

Acknowledgments

Supported by EPSRC grants EP/R030650/1, EP/T019603/1, EP/R030642/1, and EP/T019034/1.

References

[1] D. Raggi, G. Stapleton, A. Stockdill, M. Jamnik, G. Garcia Garcia, P. C.-H. Cheng, Representational systems theory: A unified approach to encoding, analysing and transforming representations, manuscript submitted for publication (2022).

[2] D. Raggi, G. Stapleton, A. Stockdill, M. Jamnik, G. Garcia Garcia, P. C.-H. Cheng, Inference and transformation in representational systems theory, in preparation for submission (2022).

[3] M. Jamnik, P. C.-H. Cheng, Endowing Machines with the Expert Human Ability to Select Representations: Why and How, Oxford University Press, 2021, pp. 355–378.

[4] P. Cheng, G. Garcia Garcia, D. Raggi, A. Stockdill, M. Jamnik, Cognitive properties of representations: A framework, in: International Conference on Theory and Application of Diagrams, Springer, 2021, pp. 415–430.

[5] D. Raggi, An implementation based on RST, https://github.com/danielraggi/rep2rep, 2022.

[6] L. Wetzel, Types and Tokens, in: E. N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, Fall 2018 ed., Metaphysics Research Lab, Stanford University, 2018.

[7] B. Huffman, O. Kunčar, Lifting and transfer: A modular design for quotients in Isabelle/HOL, in: International Conference on Certified Programs and Proofs, Springer, 2013, pp. 131–146.

[8] J. Reynolds, Types, abstraction and parametric polymorphism, in: Information Processing 83, Proceedings of the IFIP 9th World Computer Congres, 1983, pp. 513–523.

[9] C. S. Coen, A semi-reflexive tactic for (sub-) equational reasoning, in: International Workshop on Types for Proofs and Programs, Springer, 2004, pp. 98–114.

[10] D. Gentner, Structure-mapping: A theoretical framework for analogy, Cognitive science 7 (1983) 155–170.