

Modular design patterns for neural-symbolic integration: refinement and combination

Till Mossakowski¹

¹*Otto-von-Guericke-Universität Magdeburg, Germany*

Abstract

We formalise some aspects of the neural-symbol design patterns of van Bekkum et al., such that we can formally define notions of *refinement* of patterns, as well as modular *combination* of larger patterns from smaller building blocks. These formal notions have been implemented in the heterogeneous tool set (Hets), such that patterns and refinements can be checked for well-formedness, and combinations can be computed.

Keywords

design pattern, refinement, combination, ontology

1. Introduction

The integration of subsymbolic and symbolic methods in artificial intelligence, known as neural-symbolic integration, has been studied for three decades now [1, 2, 3, 4], with a recently growing interest [5, 6, 7, 8]. While a solid theoretical basis is mostly lacking [9], hundreds of specific methods and architectures have been engineered and proven to be superior to both purely symbolic and to purely subsymbolic methods. A certain structure has been brought into this plethora of methods by developing classification schemas [10, 11, 12]

We here follow neural-symbolic *design patterns* developed in [12]. They allow the description of neural-symbolic systems using small building blocks like instances, models, processes and actors in a modular way, and provide a useful visualisation of the architecture of such systems. However, these design patterns remain informal in [12]. The goal of the present work is to formalise some aspects of these design patterns, such that we can define notions of *refinement* of patterns, as well as modular *combination* of larger patterns from smaller building blocks.

Potential target audiences of our formalisation and tool support are designers of neural-symbolic systems, researchers who want to create post-hoc descriptions of such systems, as well as researchers who want to relate and systemantically arrange such systems in some kind of taxonomy.

NeSy 2022, 16th International Workshop on Neural-Symbolic Learning and Reasoning, Cumberland Lodge, Windsor, UK


✉ till.mossakowski@ovgu.de (T. Mossakowski)

🌐 <https://iks.cs.ovgu.de/~till> (T. Mossakowski)

🆔 0000-0002-8938-5204 (T. Mossakowski)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

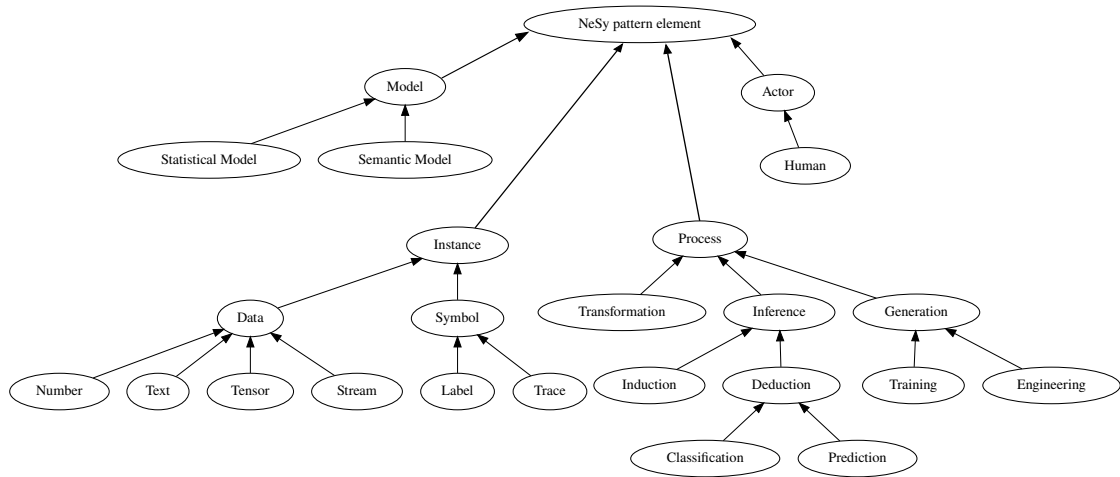


Figure 1: Ontology of pattern elements

2. Neural-symbolic design patterns

The language for design patterns describing neural-symbolic systems developed in [12] is based on a taxonomy of pattern elements that has been introduced in the appendix of [12], and that is shown in Fig. 1. An extended version with more focus on actors is presented in [13]. Note that the top class in [13] is `Boxology Taxonomy`, which we here replace by `NeSy pattern element`, which more appropriately characterises the involved objects. The classes just below the top class `NeSy pattern element` are the following:

Instance Instances are symbols or data that comprise the input or output of neural symbol systems.¹

Model Models can be hand-crafted or trained from instances (the latter is learning by induction). Existing models can be applied to instances (deduction).

Process Possible processes are training and deduction.

Actor Actors can be e.g. human beings that handcraft a model; [13] provides a more fine-grained discussion of actors.

Fig. 2 shows a simple pattern that generates a statistical model (like a neural network) from data, as well as a more complex truly neural-symbolic pattern integrating a statistical and a semantic model. Both patterns stem from [12].

3. Formalisation

We now formalise the pattern language of [12]. Patterns are some form of directed graphs. At a first glance, it seems that these graphs are bipartite, such that only edges between process

¹Ontologically speaking, these are not instances but elements of design patterns specifying that at this place of the system architecture, in the real system there will be an input or output of instances.

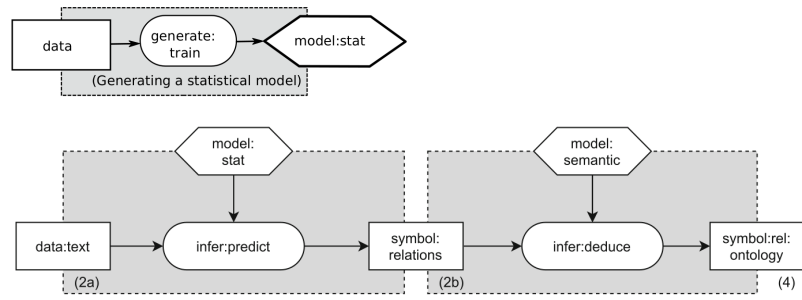


Figure 2: Two sample patterns

nodes and non-process nodes are possible. However, examples [13] show that this would be overly restrictive. Hence, we arrive at:

Definition 1. Fix an ontology (class hierarchy) of pattern elements. A neural-symbolic design pattern (NeSy pattern) is a simple directed graph, where nodes n are labeled with classes $lab(n)$ from the ontology.

While this definition is dependent on an ontology of pattern elements, it is of course desirable to achieve a standardisation here. We have formalised the ontology defined in the appendix of [12] as an OWL2 ontology in Manchester syntax at <https://ontohub.org/meta/NeSyPatterns.omn>, such that it can be used in NeSy patterns. However, for specific patterns, one might feel the need to extend this ontology with new pattern elements. Therefore, Def. 1 does not fix the ontology, but rather allows any ontology. In practice, it will be desirable to collect such ad-hoc extensions of the ontology and integrate them into a standardised ontology, whenever it seems appropriate.

During a development of NeSy patterns, one starts with rather abstract patterns, which can later be refined towards a specific systems. We formalise this using the notion of *refinement* of NeSy patterns. Refinements map patterns using a graph homomorphism. Labels can become more specific, i.e. move downwards in the class hierarchy of the ontology.

Definition 2. Given NeSy patterns P_1 and P_2 over the same ontology, a refinement $\varphi : P_1 \rightarrow P_2$ is a homomorphism of unlabelled graphs $\varphi : P_1 \rightarrow P_2$, such that for each node $n \in P_1$, $lab(\varphi(n)) \leq lab(n)$ holds in the class hierarchy of the ontology.

Fig.3 shows an abstract pattern about generating a model from instances, which is then refined in two ways: first, into a pattern where a statistical model is generated from data, and second, into a pattern where a semantic model is generated from symbols.

Patterns and refinements can be combined into *networks*:

Definition 3. A network consists of a graph with NeSy patterns as nodes and refinements as edges, showing how the patterns are interlinked. Type-correctness must hold, that is, an edge between pattern P_1 and pattern P_2 must be a refinement from P_1 to P_2 .

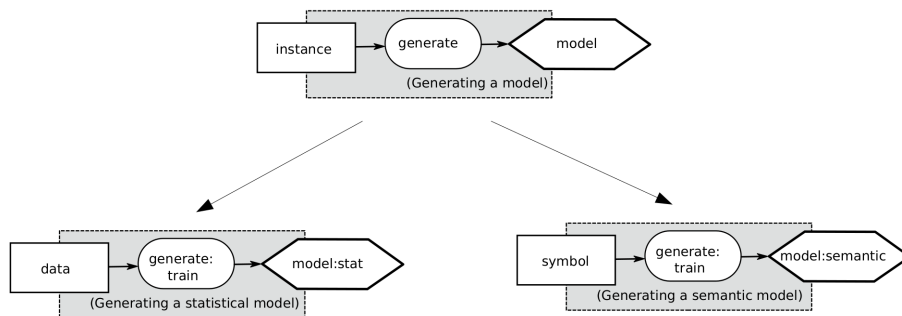


Figure 3: Two refinements of patterns

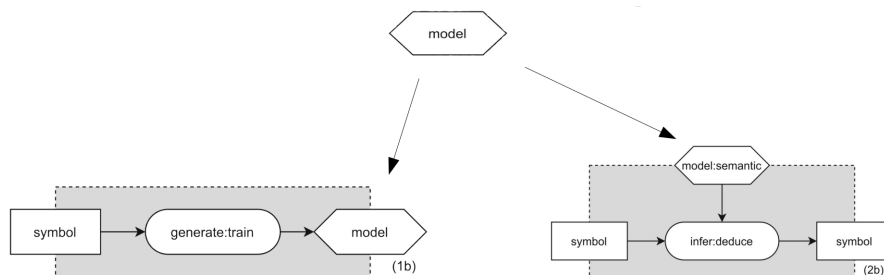


Figure 4: A network of three patterns and two refinements

Fig. 4 shows such a network. The graph homomorphisms of the refinement map the single model element of the upper pattern to the model elements of the left and the right pattern, resp. The importance of networks is that each network specifies a specific way to glue together the patterns of the network into a combined pattern.

Combination of patterns can be formally defined very succinctly as *colimits* in the sense of category theory [14]:

Definition 4. Given a network of NeSy patterns and refinements, its combination (if existing) is the colimit in the category of NeSy patterns and refinements.

For readers not familiar with category theory, we will explain this construction in more detail below. But let us first look at some example. Fig. 5 shows a combination of two patterns. The left pattern about generating a model is combined with the right pattern about inferencing using a semantic model. The two patterns are glued together at their model parts, using a very simple pattern at the top, consisting just of a Model. The combination (shown at the bottom) glues together the two input patterns. Note that the model is a semantic model here, because the infimum of Model and Semantic model in the ontology is Semantic model².

Now we give the explanation of the colimit construction in Def. 4 in elementary terms. The

²In Fig. 5, annotations model and model:semantic are used instead of the proper ontology terms. In the future, this should be unified, either by providing additional class labels in the ontology, or by adapting the figures.

construction is basically a glueing of all patterns of the network at hand, realised by a disjoint union of node sets, which is then quotiented as specified by the refinements in the network:

Proposition 5. *The colimit as specified in Def. 4 can be built as follows. Given a network, let $(P_i)_{i \in I}$ be the family of patterns of the network, and let $(\varphi_k)_{k \in K}$ be the family of refinements of the network (each coming with a source and target pattern). Let $N(P_i)$ be the set of nodes of pattern P_i . Let*

$$N = \bigsqcup_{i \in I} N(P_i)$$

be the disjoint union of all node sets, and $\nu_i : N_i \rightarrow N$ (for $i \in I$) be the injections into the disjoint union. Let \sim be the equivalence relation over N generated by

$$\nu_i(n) \sim \nu_j(\varphi_k(n)) \quad (n \in N_i)$$

for all $\varphi_k : P_i \rightarrow P_j$ in the network. Let

$$q : N \rightarrow N/\sim$$

be the factorisation of N by \sim .

The colimit pattern P has node set $N(P) = N/\sim$. Let $\mu_i : N(P_i) \rightarrow N(P)$, defined as $\mu_i = q \circ \nu_i$, be the injection of the i -th pattern into the colimit. These injections together form the so-called colimit injections. Edges in P are all pairs

$$(\mu_i(n_1), \mu_i(n_2))$$

for (n_1, n_2) an edge in pattern P_i . That is, P contains exactly those edges that are required to turn the colimit injections into graph homomorphisms.

Finally, for a node $n \in N(P)$, let $N_n = \bigcup_{i \in I} \mu_i^{-1}(n)$. Then

$$lab(n) = \inf_{m \in N_n} lab(m)$$

Now the colimit exists if the above infimum $\inf_{m \in N_n} lab(m)$ exists for all nodes $n \in N(P)$ — otherwise, the colimit is not defined. \square

Undefinedness of the colimit can arise for example if a node annotated with `Model` is refined to (a) a node annotated with `Semantic model` and (b) a node annotated with `Statistical model`. Since there is no common subclass of `Semantic model` and `Statistical model` in the ontology, the above infimum does not exist. Note that this situation can change if we change the ontology, i.e. by adding a term `Hybrid model` as a subclass of both `Semantic model` and `Statistical model`. Indeed, such a term will be very useful for specifying a pattern for logical neural networks [8].

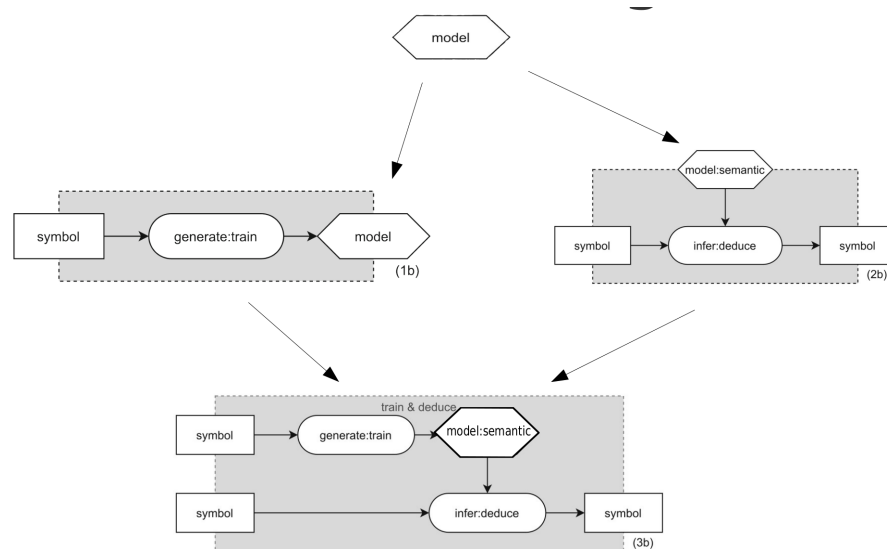


Figure 5: Combination of the network of Fig. 4.

4. Implementation

The formal notions of patterns and refinements have been implemented in the heterogeneous tool set (Hets) [15].³ Using the structuring meta language DOL [16]⁴, patterns and refinements can be written down as shown in Fig. 6. Using the declaration `data ontotHub:NeSyPatterns.omn`,⁵ each pattern refers to some ontology of pattern elements. Here, `ontotHub:NeSyPatterns` is a CURIE that abbreviates the URL <https://ontotHub.org/meta/NeSyPatterns> (using a prefix declaration). Instead, the ontology could also be specified directly by a URL, or an OWL2 ontology can be specified inline.

In the patterns, terms of the ontology have to be used exactly as they are, while in the visualised patterns, often abbreviations and/or annotations with superclasses are used. Also note that optional node identifiers can be prepended with a colon (this similar to the notation `a:C` in OWL2 ABoxes). This is important for distinguishing nodes that are annotated with the same pattern element, and for referencing nodes that have been declared earlier. E.g. the notation `d : Deduction` in the example in Fig. 6 ensures that there is only one node of type `Deduction`, and not two. The example also shows how the DOL language allows the definition of networks of patterns and refinements. Such networks can then be combined into a new pattern, in the sense of Def. 4.

Hets will check these patterns, networks and combinations for well-formedness, and the result of the combination can be computed.⁶ For the example from Fig. 6, a Hets development

³Hets is freely available under a GPL licence at <https://github.com/spechub/Hets>

⁴See also <https://dol-omg.org>

⁵The use of the `data` keyword is not related to the term `Data` of the ontology. It has historical reasons, because it is also used for linking process logics with logics for data in Hets.

⁶For details, see <https://github.com/spechub/Hets/wiki/Patterns-for-neural-symbolic-systems>

```

1 %prefix( ontohub: <https://ontohub.org/meta/> )%
  logic NeSyPatterns
3 pattern Model = data ontohub:NeSyPatterns.omn
  Model;
5 end
  pattern Train = data ontohub:NeSyPatterns.omn
7   Symbol -> Training -> Model;
  end
9 pattern SemanticDeduction = data ontohub:NeSyPatterns.omn
  Symbol -> d : Deduction -> Symbol;
11 Semantic_Model -> d : Deduction;
  end
13 refinement R1 = Model refined to Train
  end
15 refinement R2 = Model refined to SemanticDeduction
  end
17 network N =
  Train, SemanticDeduction, R1, R2
19 end
  pattern SemanticGenerateAndTrain =
21 combine N
  end

```

Figure 6: The combination of Fig. 5, formally specified in DOL

graph showing the different patterns (as nodes) and refinements (as edges) is shown in Fig. 7. By clicking on the individual nodes, one can display the individual NeSy patterns. The upper arrow in Fig. 7 is a double arrow. It is not a NeSy pattern refinement, but it rather embeds the OWL2 ontology into an intermediate NeSy pattern node, which is then used by the three declared NeSy patterns.

The modular design and re-use of NeSy patterns has been advocated in [12]. With our formalisation in DOL, we now can write down modular NeSy patterns in a precise syntax. We also have formalised all examples of [12] in a library⁷, such that system architects can re-use these patterns and build new ones on top of them easily. The implementation in Hets allows us to flatten complex modular designs and look at the resulting NeSy patterns. Fig. 8 shows one such pattern (pattern (2d) in [12]); it requires an extension of the OWL2 ontology, because the class *Embedding* has not been included in the ontology so far.

5. Conclusion and future work

We have formalised neural-symbolic design patterns using simple graphs over some ontology of neural-symbolic elements. We deliberately have not used OWL2 ABoxes or RDF for formalising design patterns. Compared to such a formulation, our formulation as simple graphs is simpler,

⁷See <https://github.com/spechub/Hets-lib/tree/master/NeSy>

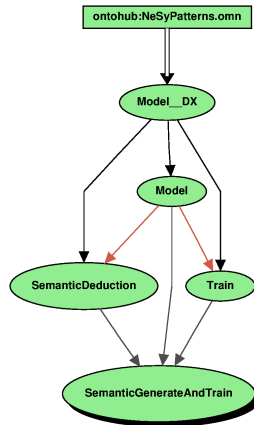


Figure 7: Hets development graph for the example of Fig. 6.

```

%prefix( ontohub: <https://ontohub.org/meta/> )%
2 logic NeSyPatterns
pattern Embedding =
4 data { ontohub:NeSyPatterns.omn
      then Class Embedding SubClassOf: Transformation }
6 Symbol -> e:Embedding -> Data; Semantic_Model -> e:Embedding;
end
  
```

Figure 8: Pattern (2d) of [12], formally specified in DOL

clearer and more concise to write. This also means that the suitable notions of refinements and colimit are simpler than those for OWL2 ABoxes or RDF. Reasoning about refinement currently is done by Hets' static analysis, which checks the inequality of Def. 2. That said, we will provide a translation of our pattern language into OWL2 ABoxes. We will use the relations (object properties) `providesInput` and `hasOutput` to express edges in the graphs that comprise design patterns. Our notation `a : Symbol -> b : Training -> c : Model` would be translated into

```

1 a : Symbol
  providesInput(a,b)
3 b : Training
  hasOutput(b,c)
5 c : Model
  
```

which despite the use of concise description logic notation is still far more verbose (and OWL2 syntax would be even more verbose).⁸

⁸The idea to formalise everything below `Process` as relation (object property), such that the above situation can be characterised by one triple `b(a,c)`, is not ontologically valid, because processes are not relations. Moreover, in [13], links between processes are used, which could not be easily represented in this schema, while we can represent these, using a further relation like `throughput`.

The formalisation of neural-symbolic patterns paves the way for several useful developments. First, the use of a formal syntax for patterns and of a formal ontology for pattern elements leads to precision and standardisation. It would be useful to develop the ontology of pattern elements as a joint community effort, such that it can be referenced in patterns. Note that the DOL language allows the local extension of the ontology, which (as we expect) will often be needed. Moreover, DOL also allows the parallel refinement of a pattern and its ontology, such that patterns written over different ontologies can be refined and combined, too. Of course, for the sake of unification, such local ontology extensions could and should be later integrated into the ontology, if found useful by the community.

Secondly, the ontology could also be used to impose constraints on patterns. For example, using the above sketched translation to an OWL2 ABox, we could state that only machine learning models can be trained by adding the axiom

$$\exists \text{hasOutput}^{-1}.\text{Training} \sqsubseteq \text{'Statistical Model'}$$

This axiom can be used to reason about patterns, with the outcome that e.g. `Statistical Model` need to be refined into `Model`. A further axiom stating disjointness of `Statistical Model` and `Semantic Model`⁹ would make patterns featuring a training of a semantic model inconsistent, which can be found by OWL2 reasoners. Hence, axioms can help to ensure the internal consistency of patterns.

As noted in [12], patterns could be also used to specific and build real neural-symbolic systems in a modular way. A step towards this goal would be to equip pattern elements with signatures. For symbols, this would be a first-order signature, such that symbols could be represented as terms over that signature. For data tensors, one would specify their dimensions. Such signatures could then be combined into more complex signatures for the specification of models and processes. Once such signatures have been provided, the next step will be to use logical languages like the Hoare-style logic of [17] for the axiomatic specification and verification of neural-symbolic systems.

Acknowledgements The author wants to thank Fabian Neuhaus for useful feedback and discussions, the anonymous reviewers for useful suggestions and Mihai Codescu and Björn Gehrke for working on the Hets implementation.

References

- [1] G. G. Towell, J. W. Shavlik, Extracting refined rules from knowledge-based neural networks, *Mach. Learn.* 13 (1993) 71–101. URL: <https://doi.org/10.1007/BF00993103>.
- [2] A. S. d’Avila Garcez, K. Broda, D. M. Gabbay, *Neural-symbolic learning systems - foundations and applications*, *Perspectives in neural computing*, Springer, 2002. URL: <https://doi.org/10.1007/978-1-4471-0211-3>.
- [3] B. Hammer, P. Hitzler (Eds.), *Perspectives of Neural-Symbolic Integration*, volume 77 of *Studies in Computational Intelligence*, Springer, 2007.

⁹However, note that such an axiom is debatable, because there are hybrid models like Logical neural networks [8] that could be considered to be both statistical and semantic.

- [4] A. S. d'Avila Garcez, L. C. Lamb, D. M. Gabbay, *Neural-Symbolic Cognitive Reasoning*, Cognitive Technologies, Springer, 2009. URL: <https://doi.org/10.1007/978-3-540-73246-4>.
- [5] P. Hitzler, M. K. Sarker (Eds.), *Neuro-Symbolic Artificial Intelligence*, Studies on the Semantic Web, IOS Press, 2022.
- [6] S. Badreddine, A. d'Avila Garcez, L. Serafini, M. Spranger, Logic tensor networks, *Artif. Intell.* 303 (2022) 103649. URL: <https://doi.org/10.1016/j.artint.2021.103649>.
- [7] J. Chen, P. Hu, E. Jiménez-Ruiz, O. M. Holter, D. Antonyrajah, I. Horrocks, Owl2vec*: embedding of OWL ontologies, *Mach. Learn.* 110 (2021) 1813–1845. URL: <https://doi.org/10.1007/s10994-021-05997-6>.
- [8] R. Riegel, A. G. Gray, F. P. S. Luus, N. Khan, N. Makondo, I. Y. Akhalwaya, H. Qian, R. Fagin, F. Barahona, U. Sharma, S. Ikbal, H. Karanam, S. Neelam, A. Likhyan, S. K. Srivastava, Logical neural networks, *CoRR abs/2006.13155* (2020). URL: <https://arxiv.org/abs/2006.13155>. arXiv:2006.13155.
- [9] F. van Harmelen, Preface: The 3rd AI wave is coming, and it needs a theory, in: P. Hitzler, M. K. Sarker (Eds.), *Neuro-Symbolic Artificial Intelligence*, Studies on the Semantic Web, IOS Press, 2022, pp. v–vii.
- [10] L. von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy, M. Walczak, J. Garcke, C. Bauckhage, J. Schuecker, Informed Machine Learning - A Taxonomy and Survey of Integrating Knowledge into Learning Systems, *IEEE Transactions on Knowledge and Data Engineering* (2021). URL: <http://arxiv.org/abs/1903.12394>. arXiv:1903.12394.
- [11] L. D. Raedt, S. Dumancic, R. Manhaeve, G. Marra, From statistical relational to neuro-symbolic artificial intelligence, in: C. Bessiere (Ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI 2020, ijcai.org, 2020, pp. 4943–4950. URL: <https://doi.org/10.24963/ijcai.2020/688>.
- [12] M. van Bekkum, M. de Boer, F. van Harmelen, A. Meyer-Vitali, A. ten Teije, Modular design patterns for hybrid learning and reasoning systems, *Appl. Intell.* 51 (2021) 6528–6546. URL: <https://doi.org/10.1007/s10489-021-02394-3>.
- [13] A. Meyer-Vitali, W. Mulder, M. H. T. de Boer, Modular design patterns for hybrid actors, *CoRR abs/2109.09331* (2021). URL: <https://arxiv.org/abs/2109.09331>. arXiv:2109.09331.
- [14] J. Adámek, H. Herrlich, G. Strecker, *Abstract and Concrete Categories*, Wiley, New York, 1990.
- [15] T. Mossakowski, C. Maeder, K. Lüttich, The heterogeneous tool set, Hets, in: O. Grumberg, M. Huth (Eds.), *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 519–522. URL: https://doi.org/10.1007/978-3-540-71209-1_40.
- [16] T. Mossakowski, M. Codescu, F. Neuhaus, O. Kutz, The distributed ontology, modeling and specification language – DOL, in: A. Koslow, A. Buchsbaum (Eds.), *The Road to Universal Logic*, volume 2, Birkhäuser, 2015, pp. 489–520. URL: <http://www.springer.com/gp/book/9783319101927>.
- [17] X. Xie, K. Kersting, D. Neider, Neuro-symbolic verification of deep neural networks, in: L. D. Raedt (Ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, IJCAI 2022, Vienna, Austria, 23-29 July 2022, ijcai.org, 2022, pp. 3622–3628. URL: <https://doi.org/10.24963/ijcai.2022/503>.