`adaptexts`: A Framework for Adapters of Formal Contexts

Tobias Hille*[1,2] and Lowejatan Noori[1]

[1] Knowledge & Data Engineering Group, University of Kassel, Kassel, Germany
[2] Interdisciplinary Research Center for Information System Design, University of Kassel, Kassel, Germany
hille@cs.uni-kassel.de, l.noori@uni-kassel.de

Abstract. Formal Concept Analysis (FCA) provides a rigorous framework for transforming raw data into formal contexts, yet acquiring such contexts from heterogeneous, real-world sources remains labor-intensive. The `adaptexts` *Python* package[1] introduces a generic adapter architecture that allows for custom conversion of diverse data sources. By encapsulating common properties (e.g., determinism, versionability, sortability) and offering a caching mechanism for expensive transformations, this package streamlines the preparation of experimental data and paves the way for reusable, composable "tools" that operate on sets of contexts. We include implementations of the adapter interface to obtain formal contexts from Git repositories, integer-partition, and UCI-ML datasets. This work outlines the design, current implementation, and future directions of the package, highlighting a path towards an extensible platform for FCA-based data analysis.

Keywords: Formal Contexts · Data Preprocessing · Python

1 Introduction

In FCA, data is represented in a strict and uniform structure known as a formal context. However, in most practical scenarios, data is rarely available in such a well-defined form. Typical examples include datasets originating from real-world measurements, integer partition constructions, the UCI-ML repository, and knowledge bases such as WikiData. In research practice, data sets are therefore often manually transformed into formal contexts. Likewise, methods operating on these data frequently overlap, resulting in a recurring set of operators that are repeatedly reimplemented or copied across projects. To reduce this repetitive effort, an *adapter* can provide a unifying solution: one that accepts arbitrary data sources (e.g., Git repositories, open datasets, or generated structures), automates the necessary transformations, and minimizes manual post-processing with *tools* that act on sets of formal contexts. The `adaptexts` package written in *Python* in the current state introduces a foundation for adapters

* Corresponding author
[1] `https://codeberg.org/thille/adaptexts`

targeting heterogeneous data sources, together with reference implementations. The main task of an adapter is to load raw data from a heterogeneous source and transform it into one or more formal contexts, following a uniform iterator-based interface. In this work, we outline the expected design, behaviour, properties of such adapters, and the current state of the package. To address the well-known challenges of adopting tools, we have deployed a public adapter browser[2] that catalogues all available adapters and their properties. This enables discoverability and community contribution while keeping technical requirements to a minimum.

Related Work In the broader machine learning landscape, generic dataset repositories provide centralised, programmatic access to large collections of data. Yet such libraries offer neither FCA-specific semantics nor automated conversion into formal contexts, leaving users to perform manual transformations on their own. Within the FCA community, tools typically serve small groups of developers, each maintaining their own data repositories as isolated silos. For instance, the fcatools/contexts repository [7] collects formal contexts, yet does not offer a standardised programmatic interface in a high-level programming language. Similarly, the FCA Tools Bundle project [13] offers a web-based collection of FCA utilities and community-contributed contexts, but again lacks a standardised API for programmatic integration into research pipelines. The fragmentation of such isolated repositories highlights the need for a unified adapter architecture that allows research projects to access existing data without reimplementing conversion logic for each source.

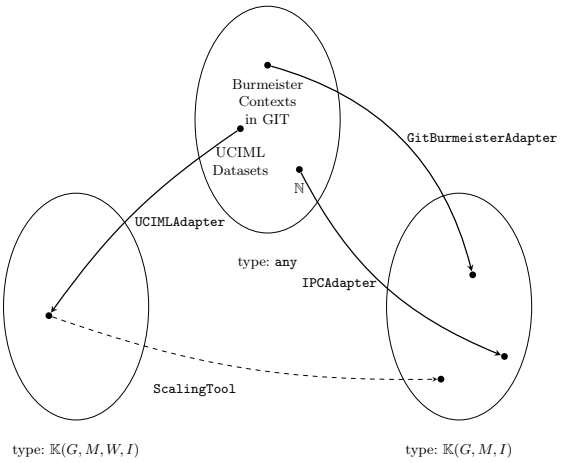

Fig. 1: An illustrative example of using adapters and tools. The mentioned adapters and tools are fully implemented.

[2] `https://al-noori.github.io/adaptexts-web/`

2 Adapter and Tools Design

Formally, an adapter is a computable mapping $\alpha : S \to \mathcal{K}$, where S is a data source space and $\mathcal{K}$ is the space of formal contexts. Each α in the set of all adapters $\mathcal{A}$ yields contexts via a Python *iterator* interface. In general, an adapter is created with a source of any type. The adapter should only convert any type of data into formal contexts. Depending on the data, a formal context can be derived directly or by scaling a many-valued formal context, due to e.g. numerical features.

Regardless of the input, an adapter is expected to behave like a Python *iterator* and yield various unary formal contexts. The reasoning behind this is to minimize specialized data handling outside the initial data transformation. A fully implemented adapter must always state properties that give insight on its inner workings and outside behavior on a high abstraction level. We list a small collection of properties worth knowing for the end user. For the reference adapters, most of these properties are trivially true, but that could look different for random generated data (e.g. *determinism*) or recursively defined data (e.g. *statelessness*). Furthermore, we include caching mechanisms to some of our implementations, as those help with user experience when relying on expensive computations for building the formal context.

Formally, a tool is a mapping $\tau : \mathcal{K}_A \to \mathcal{K}_B$ between spaces of formal contexts of potentially different types (e.g., unary, many-valued, incomplete). We require tools to be composable: for tools $\tau_1 : \mathcal{K}_A \to \mathcal{K}_B$ and $\tau_2 : \mathcal{K}_B \to \mathcal{K}_C$, the composition $\tau_2 \circ \tau_1$ is well-defined. The aim is that tools provide easy access to common operations needed when working with formal contexts, and give building blocks for composable data pipelines in general. One design idea for now is that tools might work similar to python's built-in `iterable` functions, like `zip` or `filter`, and operate on an iterable collection of formal contexts, and yield objects of context (a sub-)type as well.

2.1 Adapter Properties

We argue that from a user perspective a small collection of properties that categorize the internal adapter behavior is useful. It gives users the possibility to group and select available adapters based on use-case. In the following, we list the properties and describe possible use-cases.

Determinism ensures reproducible experiments and predictable behaviour. This can be distinguished into source determinism (the same source produces the same raw data) and adapter determinism, e.g. avoiding iterations over sets within the conversion to formal contexts.

Sortability refers to the adapter's ability to return formal contexts in a consistent, well-defined order, rather than relying on whatever order happens to come from the underlying storage or source. This is tightly connected to determinism.

Versionability is the property of accessing a specific temporal state of the source data. This is most crucial for experiments, as these rely on reproducibility. In GitHub repositories, revisions grant detailed access to a chosen state, whereas for example an adapter for WikiData might only be able to provide publicly available snapshots of the entire database.

Metadata availability can indicate whether an adapter can provide readable metadata (e.g. in JSON or XML format) for every formal context they yield. Metadata can act as a channel for discovery, filtering, and reproducible selection of formal contexts, and allows users and (automated) tools to work with additional information without loading the context first.

Statelessness is a design goal that simplifies reasoning about internal behavior of the implemented adapter functionality and makes them safer to run in concurrent environments.

3 Implementations of Adapters and Tools

The package provides the data structures `Context` and `ManyValuedContext` as the basis for implementing adapters. Their implementation follows their usual mathematical definitions [4, Def 20, Def 34]. This means that we encode a formal context as 3-tuple (G, M, I), where G is a set of objects, M a set of attributes, and $I \subseteq G \times M$ the incidence relation. Moreover, a many-valued context combines (G, M, W, I), where W denotes the set of values and $I \subseteq G \times M \times W$ assigns exactly one value to each object–attribute pair.

We provide a collection of various `FormatMixins`[3] that allows classes equipped with it to be instantiated and serialized to common storage formats, for example *ctx,json,pickle* and *pandas.DataFrame*.

Adapters have to be fully implemented subclasses of the provided abstract `AdapterInterface`. To extend functionality without repeated boilerplate code, we provide the two powerful, yet composable `CacheMixin` and `IterableMixin` that extend the given adapter to give direct access to the managed underlying formal contexts via an unique key. The CacheMixin is designed to allow the easy intermediate storage of formal contexts that were obtained by expensive processed from the original data source. The IterableMixin expects an adapter method `_get(key:Any)` that returns the actual context given its key.

During implementation and initial usage, we observe that most data sources in the world of FCA are actually stored as simple files in directory trees. Thus we included `FileTreeMixin` that allows for automatic path dependent context key generation with detailed inclusion/exclusion rules. The `DirectoryAdapter` is equipped with both the FileTreeMixin and the IterableMixin and gives developers and researchers a straightforward base for implementing new adapters.

[3] `https://wiki.c2.com/?MixIn`

The implementation of the `GitAdapter` (as subclass of the DirectoryAdapter) is an attempt to come as close as possible to an ideal universal but fully implemented adapter for a wide range of sources. As implementing a dedicated adapter for every possible source is obviously a nontrivial task, we include the GitAdapter as a general foundation for adapting data from git repositories into formal contexts. Developers are encouraged to create custom adapters as subclasses. As the GitAdapter has access to the functionality of the FileTreeMixin (through its parent class DirectoryAdapter), any class inheriting from it only has to decide on what files to consider. By overwriting the required but already provided `_get` method mentioned earlier, file handling for uncommon types is possible. For this adapter, *versionability* is inherently provided through git revisions.

Furthermore, the *adaptexts* package provides the `GitBurmeisterAdapter` as a specialized GitAdapter, which expects contexts to be stored as Burmeister-formatted files in the source repository. The fully implemented and instantiable adapters are explained in the following subsections.

3.1 IPC Adapter

The set of all partitions of $n \in \mathbb{N}$ can be endowed with a natural ordering [4, p. 54]. The IPC Adapter is conceptually a mapping of natural numbers to their integer partition context. By construction, only that input is required. The actual computation of the integer partition is abstracted away from the user, but in this case is done by the adapter itself. For example, the context of integer 8 can be simply obtained via

```
context = IPCAdapter(range(9)).get(8)
```

The adapter is using the CacheMixin, thus calling the `get` method invokes computing the integer context only if it is not already cached. Already computed contexts are stored locally in Burmeister format.

3.2 RWC Adapter

The RWC adapter links to the `fcatools/contexts` repository [7][4] and is a minimal extension of the GitBurmeisterAdapter. The class provides the additional ability of serving metadata associated with the context (by key). Given its inheritance hierarchy, the managed contexts can be directly accessed as follows:

```
context = RWCAdapter().get("contexts/music_en.ctx")
```

Internally, the adapter is initialized with the repository URL, clones the repository and converts the Burmeister format via `from_burmeister(content: str)` into a context object.

[4] `https://github.com/fcatools/contexts`

3.3 UCI-ML Adapter

This adapter links to datasets of the UCI-ML repository[9] via functionality originally provided by the `ucimlrepo` package[5][6]. A typical data set from that collection consists of finitely many samples with integer, continuous, or categorical features, which can naturally be interpreted as a many-valued context. The adapter yields many-valued contexts at its core, and handles downloading, caching, metadata based filtering and preprocessing.

To make working with the adapter easier, we included a "get_context" method that can handle both the official data set name as well as its data set ID. An exemplary usage with an output via `context.to_df()` is as follows:

```
uciml_filter = UCIMLFilter(include_missing=False, max_features=1000,
                           max_instances=10000)
context = UCIMLAdapter(uciml_filter=uciml_filter).get_context("Heart
                            Disease").to_df()
```

	[age]	[sex]	...
0	63	1	...
1	67	1	...
2	67	1	...
3	37	1	...
4	41	0	...
..	..	..	...

[303 rows × 14 columns]

3.4 ScalingTool

The `ScalingTool` provides the functionality of applying standard scales to a many-valued context, automatically if necessary. Behind the scenes, it applies the scaling procedures available through `conexp-clj` [6]. This is made possible by using a simple REST API client written in Python provided by the first author[7] to access the conexp-clj library. As mentioned above, the Context and ManyValuedContext classes are provided with standard FormatMixins. Additionally, we include a mixin that allows for easier transformation in the data classes used by the conexp-clj-py package.

By default, the ScalingTool contains a small subroutine to infer a suitable scaling automatically. If automatically generated standard scales would "blow up" the context too much through scaling (or if more control is required) it is also possible to pass a custom scale map to the ScalingTool, i.e. a dictionary mapping attributes to scales and optional specifications of the form:

[5] `https://github.com/uci-ml-repo/ucimlrepo`

[6] As the ucimlrepo project is unfortunately abandoned but has a suitable license, we reimplemented and extended the functionality in the presented package.

[7] `https://codeberg.org/thille/conexp-clj-py`

```
{
  "att_1": {"name": "interordinal-scale"},
  "att_2": {"name": "ordinal-scale"},
  "att_3": {"name": "interordinal-scale", "thresholds": [3, 6, 8]},
  "att_4": {"name": "biordinal-scale", "n": 4},
  ⋮
}
```

4 Planned Extensions and Outlook

We presented the package `adaptexts`, its design choices, and the properties that characterize adapters within the framework. The package is currently in an early alpha version; tools beyond the ScalingTool remain conceptual. Overall, `adaptexts` is intended as an open foundation rather than a closed framework.

Contextual Extensions. Beyond binary contexts, the planned support for incomplete contexts is motivated by the growing prevalence of real-world datasets with missing incidence data, a theme that has also been explored in knowledge acquisition research [8]. Extending the framework to handle incomplete, polyadic, and fuzzy contexts will enable robust analysis in the presence of missing or partially observed data. This would allow transformations from these richer context spaces into unary formal contexts, giving users flexible preprocessing options for different analytical tasks.

Usability and Ecosystem Integration. A registry system is envisioned, supporting both global and local scopes for sharing and including adapters and tools, to facilitate modular workflows and community-driven extensions. A full integration into the scikit-learn ecosystem is also planned by aligning the implementation with its API conventions, enabling seamless interoperability with existing machine learning pipelines [1].

Planned Adapters. Several adapters are planned for widely used data formats and repositories. These include interfaces for KONECT and Pajek datasets[10,12], contexts from the FCA Tools Bundle [13], and synthetic data generators such as Dirichlet-based methods for benchmarking [3]. A dedicated adapter for OEIS-sequence data is under consideration [5], as well as an adapter targeting Zenodo archives, exemplified by the DimDraw user study [2].

Planned Tools. Planned tools include imputation methods for handling missing values in contexts, and similarity-based filtering to support efficient deduplication when multiple adapters provide overlapping datasets.

The package will keep evolving the adapter catalogue and introducing composable tools, while keeping adapter development accessible to end users and contributors.

References

1. Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., Varoquaux, G.: API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning. pp. 108–122 (2013)
2. Dürrschnabel, D., Hanika, T., Stumme, G.: Dataset of user study for dimdraw (Oct 2020), https://doi.org/10.5281/zenodo.4075207
3. Felde, M., Hanika, T.: Formal context generation using dirichlet distributions. In: Graph-Based Representation and Reasoning - 24th International Conference on Conceptual Structures, ICCS 2019, Marburg, Germany, July 1-4, 2019, Proceedings. pp. 57–71 (2019)
4. Ganter, B., Wille, R.: Formal Concept Analysis. Springer Nature Switzerland, Cham, Switzerland (2024)
5. Gauvrit, N., Delahaye, J.P., Zenil, H.: Sloane's Gap. Mathematical and Social Factors Explain the Distribution of Numbers in the OEIS (2011)
6. Hanika, T., Hirth, J.: Conexp-clj - A research tool for FCA. In: Cristea, D., Ber, F.L., Missaoui, R., Kwuida, L., Sertkaya, B. (eds.) Supplementary Proceedings of ICFCA 2019 Conference and Workshops, Frankfurt, Germany, June 25-28, 2019. CEUR Workshop Proceedings, vol. 2378, pp. 70–75. CEUR-WS.org (2019)
7. Hanika, T., Jäschke, R.: A Repository for Formal Contexts. In: Conceptual Knowledge Structures, pp. 182–197. Springer, Cham, Switzerland (Aug 2024)
8. Holzer, R.: Knowledge acquisition under incomplete knowledge using methods from formal concept analysis: Part II. Fundam. Informaticae **63**(1), 41–63 (2004)
9. Kelly, M., Longjohn, R., Nottingham, K.: The UCI Machine Learning Repository. https://archive.ics.uci.edu (2023), accessed: 2026-01-23
10. Kunegis, J.: Handbook of Network Analysis [KONECT – the Koblenz Network Collection] (2014), https://github.com/kunegis/konect-handbook/blob/master/konect-handbook.pdf
11. Noori, L., Hille, T.: adaptexts — dataset browser. https://al-noori.github.io/adaptexts-web/ (2026), accessed: 2026-05-29
12. de Nooy, W., Mrvar, A., Batagelj, V.: Exploratory Social Network Analysis with Pajek: Revised and Expanded Second Edition. Cambridge University Press, New York (2011)
13. Savić, D.: FCA Tools Bundle. https://fca-tools-bundle.com (2023), accessed: 2026-05-29