# BenchmarX

Anthony Anjorin
Technische Universität Darmstadt
anthony.anjorin@es.tu-darmstadt.de

Alcino Cunha
HASLab / INESC TEC and Universidade do Minho
alcino@di.uminho.pt

Holger Giese
Hasso-Plattner-Institut
holger.giese@hp.uni-potsdam.de

Frank Hermann
Université du Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust
frank.hermann@uni.lu

Arend Rensink
University of Twente
arend.rensink@utwente.nl

Andy Schürr
Technische Universität Darmstadt
andy.schuerr@es.tu-darmstadt.de

## ABSTRACT

Bidirectional transformation (BX) is a very active area of research interest. There is not only a growing body of theory, but also a rich set of tools supporting BX. The problem now arises that there is no commonly agreed-upon suite of tests or benchmarks that shows either the conformance of tools to theory, or the performance of tools in particular BX scenarios. This paper sets out to improve the state of affairs in this respect, by proposing a template and a set of required criteria for benchmark descriptions, as well as guidelines for the artifacts that should be provided for each included test. As a proof of concept, the paper additionally provides a detailed description of one concrete benchmark.

## Categories and Subject Descriptors

D.4.8 [**Software Engineering**]: Performance—*Measurements*

## General Terms

Measurement

## Keywords

bidirectional transformations, benchmark, tools, comparison

## 1. INTRODUCTION AND MOTIVATION

Bidirectional transformations (BX) are required to maintain the consistency of related artifacts modified by concurrent engineering activities [2]. This task is relevant in multiple domains and is an active research focus in various communities [2]. Although the first theoretical foundations for BX have been laid and there are a number of tools that already support BX to a certain extent [9], it is difficult for non-developers to discern the exact capabilities of each BX tool and effectively compare it with others.

What is missing is a collection of *benchmarks*, which can be used to identify the strengths and weaknesses of different tools and their respective approaches. Such a benchmark suite for BX will not only clarify the state of the art and current limitations of BX tools, but also drive further development and cross-fertilization between the different BX sub-communities and tool developers.

Our goal in this paper is to improve the current situation by proposing a template for BX benchmarks. Specifically, we:

1. Provide a precise definition for a *BX benchmark*, identifying a set of required properties that distinguish a BX benchmark from a mere BX example.
2. Identify a *feature matrix* similar to that given in [12], which is to be used to classify BX benchmarks.
3. Present, as a *proof of concept*, a simple but concrete benchmark that adheres to our proposed template.

## 2. STRUCTURE OF A BX BENCHMARK

According to the dictionary,[1] a *benchmark* is a *standardised problem or test that serves as a basis for evaluation or comparison*. The aim of *benchmarking* is to systematically assess and, if possible, measure a set of features of a system under different, precisely defined, and reproducible circumstances [12]. Based on the respective results on a benchmark, different systems can be compared and evaluated based on pertinent system characteristics. In this section we collect the characteristics needed for a BX benchmark to make such assessment and measurement possible.

### 2.1 Prerequisites

In order to precisely define different aspects and properties of bidirectional transformations in this paper, we first provide some detailed formalisations and general requirements.

A *bidirectional transformation* consists of the following:

- A *left language* $\mathcal{L}^{\mathsf{L}}$, a *right language* $\mathcal{L}^{\mathsf{R}}$, and a binary *consistency relation* $\mathcal{C}$ over both languages that specifies which pairs of left and right models are consistent. It is difficult to escape the directional cultural bias here, though we try to avoid equating "left" with "source" and "right" with "target".
- Sets of possible left updates $U^{\mathsf{L}}$ and right updates $U^{\mathsf{R}}$. Each update $u$ is a mapping from models to corresponding updated models. The application of an update to a single model is given by a *change* $\delta$ (we shall state exactly what this is in a moment), that specifies how a model is modified to a changed model. The sets of possible changes to the left and right models are denoted by $\Delta^{\mathsf{L}}$ and $\Delta^{\mathsf{R}}$, respectively.

[1] Merriam-Webster, 2013

- Propagation functions $\overrightarrow{\mathcal{C}}\colon \Delta^{\mathsf{L}} \times \mathcal{L}^{\mathsf{R}} \to \Delta^{\mathsf{R}}$ and $\overleftarrow{\mathcal{C}}\colon \Delta^{\mathsf{R}} \times \mathcal{L}^{\mathsf{L}} \to \Delta^{\mathsf{L}}$, commonly called *forward* and *backward* propagation. The forward propagation $\overrightarrow{\mathcal{C}}$ takes a change $\delta$ on the left-side together with a right model (consistent with the left model that is the source of $\delta$), and returns a change on the right-side to be applied to that model. The same applies dually for the backward propagation $\overleftarrow{\mathcal{C}}$.

We will equate languages with their extensions, being sets of *models*; hence we can regard $\mathcal{C} \subseteq \mathcal{L}^{\mathsf{L}} \times \mathcal{L}^{\mathsf{R}}$ as a binary relation between left models and right models. Consistency of a left-model $M^{\mathsf{L}} \in \mathcal{L}^{\mathsf{L}}$ and a right-model $M^{\mathsf{R}} \in \mathcal{L}^{\mathsf{R}}$ will be denoted by $M^{\mathsf{L}}\ \mathcal{C}\ M^{\mathsf{R}}$. In some scenarios, the consistency relation $\mathcal{C}$ may only be admitted if there is further evidence of consistency, for instance in the form of an element-to-element mapping between related models; this can be captured by defining $\mathcal{C}$ as the union of a set of *indexed* binary relations $\mathcal{C}_\varphi$, where $\varphi$ is the object that captures the evidence — for instance, a binary relation between the elements of the left model and those of the right model.

Members of the sets of left updates $U^{\mathsf{L}}$ and right updates $U^{\mathsf{R}}$ are partial functions from models to models. For instance, a left update $u^{\mathsf{L}} \in U^{\mathsf{L}}$ is a partial function from $\mathcal{L}^{\mathsf{L}}$ to $\mathcal{L}^{\mathsf{L}}$. We reserve the word *change* (or *delta*) for the pairs of models that are the extension of an update; i.e., if $u(M_1) = M_2$ for some update $u$ then $(M_1, M_2)$ is a change. Like pairs of consistent models, in some scenarios a change may have additional information about the relation between the elements of source and target model; in such a situation, the change will be a triple $(M_1, M_2, \mu)$ for some object $\mu$ that encodes the additional information. In general, a change $\delta$ will always have a well-defined source model $\mathrm{src}(\delta)$ and target model $\mathrm{tgt}(\delta)$. The set of all allowed left- [right-] changes is denoted by $\Delta^{\mathsf{L}}$ [$\Delta^{\mathsf{R}}$]. A single change $\delta$ may be taken as a kind of degenerate update $u$, applicable only to $\mathrm{src}(\delta)$ and yielding $\mathrm{tgt}(\delta)$.

The general requirement for $\overrightarrow{\mathcal{C}}$ is that for all $\delta^{\mathsf{L}} \in \Delta^{\mathsf{L}}$ and $M^{\mathsf{R}} \in \mathcal{L}^{\mathsf{R}}$, such that $\mathrm{src}(\delta^{\mathsf{L}})\ \mathcal{C}\ M^{\mathsf{R}}$:

$$M^{\mathsf{R}} = \mathrm{src}(\overrightarrow{\mathcal{C}}(\delta^{\mathsf{L}}, M^{\mathsf{R}})) \tag{1}$$

and analogously for $\overleftarrow{\mathcal{C}}$. These requirements are usually known as *incidence laws* in delta-based BX frameworks [3]. In the simplest case, where $\Delta = \mathcal{L} \times \mathcal{L}$ is the set of all possible pairs of models, (1) can be trivially satisfied. It should be noted that, when $\mathcal{C}$ is left-unique and left-total (see below), then the target of $\overrightarrow{\mathcal{C}}(\delta^{\mathsf{L}}, M^{\mathsf{R}})$ is uniquely defined by the target of $\delta^{\mathsf{L}}$; and dually for the other direction. A special case of (say forward) propagation is where $\delta^{\mathsf{L}}$ is the change from some initial, empty model $\perp \in \mathcal{L}^{\mathsf{L}}$ into our left model of interest $M^{\mathsf{L}}$ (often referred to as a *batch* transformation).

How forward and backward propagation are defined is up to the benchmark in question. For instance, if there is additional *evidence* $\mathcal{C}_\varphi$ associated with the consistency of $M^{\mathsf{L}}$ and $M^{\mathsf{R}}$, then $\mathcal{C}_\varphi$ might play a role in the definition of $\overrightarrow{\mathcal{C}}$ and $\overleftarrow{\mathcal{C}}$ (and be required as input).

## 2.2 Definition of a BX benchmark

The following definitions are adapted from [12] and adjusted as required for BX benchmarks:

A *BX scenario* is a broad application field that can be clearly characterized as requiring BX. Examples include model synchronization, tool integration, and round-trip engineering.

A *BX benchmark* is a bidirectional transformation that serves as an incarnation of a BX scenario, with the following properties:

1. A precise, *executable* definition of the consistency relation $\mathcal{C}$, which can be used as an oracle to decide if a left model $M^{\mathsf{L}} \in \mathcal{L}^{\mathsf{L}}$, and a right model $M^{\mathsf{R}} \in \mathcal{L}^{\mathsf{R}}$ are consistent (i.e., $M^{\mathsf{L}}\ \mathcal{C}\ M^{\mathsf{R}}$).
2. An explicit definition of data (input models) for the transformation, or a *generator* that can be used to produce the required models.
3. A set of precisely defined updates for certain input models.
4. A set of *executable* metric definitions for what is to be measured / assessed by the benchmark.
5. Finally, as it is of no interest to measure arbitrary and irrelevant features, a benchmark should represent a *useful* transformation that covers aspects that are indeed relevant in the corresponding BX scenario.

A BX benchmark consists of several *test cases*, each of which is a complete, deterministic, but parametric specification fixing all details required for executing the corresponding measurements. Every test case measures or assesses specific features.

Finally, a *run* is a test case for which all parameters (e.g., input data size) are set to concrete values.

The results of carrying out a benchmark for a specific tool are produced by executing a series of runs for the different test cases of the benchmark.

## 2.3 Classification of BX benchmarks

A BX benchmark is to be classified using a *feature matrix*, with one dimension corresponding to the different test cases of the benchmark, and the other to paradigm and tool features, discussed in the following.

### 2.3.1 Paradigm features of the benchmark

A *paradigm feature* of a BX benchmark describes a characteristic of a test case of the benchmark. We identify the following paradigm features (to be extended as required):

**Properties of the consistency relation.** The following properties purely depend on the left and right languages. While in some BX approaches, the consistency relation is derived from the specification of the BX operations, we explicitly avoid this coupling. This implies that the benchmark can be seen as a test for the correctness and completeness of the particular solution using a specific approach.

1. $\mathcal{C}$ can be *left-total*, meaning that for every left model $M^{\mathsf{L}} \in \mathcal{L}^{\mathsf{L}}$, there exists *at least one* right model $M^{\mathsf{R}} \in \mathcal{L}^{\mathsf{R}}$ such that $M^{\mathsf{L}}\ \mathcal{C}\ M^{\mathsf{R}}$; and *right-total*, which is the dual and defined analogously.
2. $\mathcal{C}$ can be *left-unique*, meaning that for every right model $M^{\mathsf{R}} \in \mathcal{L}^{\mathsf{R}}$, there exists *at most one* left model $M^{\mathsf{L}} \in \mathcal{L}^{\mathsf{L}}$ such that $M^{\mathsf{L}}\ \mathcal{C}\ M^{\mathsf{R}}$; and *right-unique*, which is the dual and defined analogously.

For those more familiar with other terminology: a left-total relation is often called *total*, right-total *surjective*, left-unique *injective*, and right-unique *functional*.

**Platform dependency.** A test case can range from *platform independent* (PI) to *platform specific* (PS), depending on the nature of its description. Note that according to our definition of a BX benchmark, data *must* be provided, but this can be, e.g., in form of a textual format for which adapters for different platforms can be provided as required (PI), as opposed to, say, Ecore XMI files (PS). A benchmark should state explicitly the platforms for which data or appropriate adapters are provided.

**Characteristics of the data domain.** Every benchmark (or individual test case) should state the characteristics of the required data domain explicitly. With this we mean (i) if the data can be represented as simple sets, trees, or graphs, and (ii) what constraints are imposed including keys, an order on elements, etc. This is important as some BX tools are specialized for, e.g., sets and do not support graph-like structures.

**Variations and extension points.** Every benchmark should state if it is a variation of an existing benchmark and make the differences as explicit as possible. The goal here is to be able to document and manage a family of different benchmarks, which are clearly related, i.e., use basically the same transformation, but either simplify or introduce additional complexity.

### 2.3.2 *Tool features measured by the benchmark*

A *tool feature* of a BX benchmark represents a tool characteristic that can be assessed and measured with a series of runs of a test case of the benchmark. We identify the following tool features (to be extended as required):

**Run time:** The run time required to execute a run can be measured in, e.g., milliseconds for a tool. Note that run time can also be measured abstractly in "actions" or "edits" depending on the used technology (e.g., a database or model repository).

**Memory Consumption:** The required memory for executing a run can be measured for a tool either in (kilo)bytes or, as for run time, in more abstract units if this makes sense.

**Scalability:** By plotting run time/memory consumption for (i) increasing input data size and (ii) increasing size of propagated changes $(U^{\mathsf{L}}, U^{\mathsf{R}})$, the scalability of a tool can be measured with respect to the varied dimension. Note that this is only feasible if the test case consists of several runs of increasing size or if a size parametric data generator is provided.

**Conformance to laws for BX approaches:** The expected behaviour of BX tools is typically assessed by checking conformance to a set of laws. Depending on the BX framework, e.g., lenses [3], Triple Graph Grammars [8], or other BX algebraic frameworks [10], various sets of laws can be formulated differently.

Given $M^{\mathsf{L}} \in \mathcal{L}^{\mathsf{L}}$, $M^{\mathsf{R}} \in \mathcal{L}^{\mathsf{R}}$, $\delta^{\mathsf{L}} \in \Delta^{\mathsf{L}}$ such that $M^{\mathsf{L}}\ \mathcal{C}\ M^{\mathsf{R}}$ and $\mathrm{src}(\delta^{\mathsf{L}}) = M^{\mathsf{L}}$, we identify the following potential laws of interest (to be extended as required):

**Correctness.** A basic law present in all frameworks is conformance of the forward and backward propagation to the consistency relation $\mathcal{C}$. *Forward correctness* is stated as:

$$\left[\overrightarrow{\mathcal{C}}(\delta^{\mathsf{L}}, M^{\mathsf{R}}) = \delta^{\mathsf{R}}\right] \implies \mathrm{tgt}(\delta^{\mathsf{L}})\ \mathcal{C}\ \mathrm{tgt}(\delta^{\mathsf{R}})$$

*Backward correctness* is defined analogously.

**Completeness.** Another interesting property to assess is whether the forward propagation succeeds in returning a change whenever at least one consistent right-model exists. *Forward completeness* can be stated as:

$$\left[\exists M_2^{\mathsf{R}}\ .\ \mathrm{tgt}(\delta^{\mathsf{L}})\ \mathcal{C}\ M_2^{\mathsf{R}}\right] \implies \overrightarrow{\mathcal{C}}(\delta^{\mathsf{L}}, M^{\mathsf{R}})\downarrow$$

Here, $\overrightarrow{\mathcal{C}}(\delta^{\mathsf{L}}, M^{\mathsf{R}})\downarrow$ means that the forward propagation is defined for the given inputs, i.e., the execution is successful and returns a change on the right domain. *Backward completeness* is defined analogously.

As an alternative to the more consensual notion of correctness proposed above, some scenarios may require some notion of compatibility of forward and backward propagation operations, e.g., the GETPUT and PUTGET laws [6], or (weak) invertibility [4]. For instance, instead of requiring the result of a forward propagation to be consistent, one may only require it to be constant under an additional round trip application of backward and forward propagations. Benchmarks may vary on the used laws to assess expected behaviour.

If the consistency relation $\mathcal{C}$ is *not* source unique, backward propagation must possibly deal with a *loss of information* as it is unclear and in general impossible to reconstruct the "expected" source only given a modified target. It is important to assess if a tool is able to use the old source to cope adequately with information loss. The situation is analogous for non target uniqueness and forward propagation. The following laws can be used to assess this:

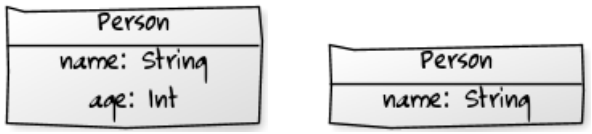


**Figure 1: Meta-models for $\mathcal{L}^{\mathsf{L}}$ (left) and $\mathcal{L}^{\mathsf{R}}$ (right)**

**Hippocraticness.** A basic property that is present in most frameworks forbids *unnecessary* changes when the left and right model are already consistent. This is most commonly known in a simplified form as *hippocraticness* [10]. *Forward hippocraticnes* can be formalized as follows (similarly for *backward hippocraticnes*):

$$\left[\mathrm{tgt}(\delta^{\mathsf{L}})\ \mathcal{C}\ M^{\mathsf{R}} \wedge \overrightarrow{\mathcal{C}}(\delta^{\mathsf{L}}, M^{\mathsf{R}}) = \delta^{\mathsf{R}}\right] \implies \mathrm{tgt}(\delta^{\mathsf{R}}) = M^{\mathsf{R}}$$

**Least Change.** A stronger requirement than hippocraticness is the *least change* property, that forbids changes that are clearly unnecessary (first introduced in [7] as the *principle of least change*). A formalization of *forward least change* is:

$$\left[\overrightarrow{\mathcal{C}}(\delta^{\mathsf{L}}, M^{\mathsf{R}}) = \delta^{\mathsf{R}}\right] \implies \nexists\, \delta_2^{\mathsf{R}}.\ \mathrm{src}(\delta_2^{\mathsf{R}}) = M^{\mathsf{R}} \wedge \mathrm{tgt}(\delta^{\mathsf{L}})\ \mathcal{C}\ \mathrm{tgt}(\delta_2^{\mathsf{R}}) \wedge \delta_2^{\mathsf{R}} <_{\Delta} \delta^{\mathsf{R}}$$

Here we assume the existence of a partial order $<_{\Delta}$ between changes that somehow measures if one change is "smaller" than another. The concrete definition of this partial order is benchmark dependent, and it may be interesting to even have different instantiations of this law for different partial orders.

## 3. AN EXAMPLE BX BENCHMARK

We presuppose primitive types **String** and **Int**. Our left and right languages are defined by the meta-models in Fig. 1. Both are extremely simple: the left language consists of *sets* of **Person**s, with name and age attributes, whereas the right language likewise consists of collections of **Person**s, in this case only with name attributes. For both languages, name is considered to be a *key* attribute, meaning that only collections of persons with distinct names are allowed. The languages are extensionally defined as sets of models with the following characteristics:

- $\mathcal{L}^{\mathsf{L}}$ is the set of finite partial functions $M^{\mathsf{L}} : \mathbf{Int} \rightharpoonup (\mathbf{String} \times \mathbf{Int})$ such that for all pairs of elements $M^{\mathsf{L}}(p_i) = (n_i, a_i)$ for $i = 1, 2$, $n_1 = n_2$ implies $p_1 = p_2$.
- $\mathcal{L}^{\mathsf{R}}$ is the set of all finite injective partial functions $M^{\mathsf{R}} : \mathbf{Int} \rightharpoonup \mathbf{String}$.

Note that models are not simply tuples of names and ages, respectively names: they include an *identity*, here taken from the set **Int**. This is essential for the benchmark, as it allows us to speak meaningfully of a given person changing his name while remaining the same entity. The actual choice of identity does not matter. Also note that, though for conciseness we have chosen to formulate models as finite partial functions from the identity set to the corresponding tuple of attributes, this is mathematically equivalent to a left-unique relation between the identity set and tuples of attributes.

**Consistency relation.** We will use $\mathsf{name}^{\mathsf{L}} : \mathbf{Int} \rightharpoonup \mathbf{String}$ [resp. $\mathsf{age}^{\mathsf{L}} : \mathbf{Int} \rightharpoonup \mathbf{Int}$] to denote the partial function mapping each person $p \in \mathrm{dom}(M^{\mathsf{L}})$ to the first [second] component of $M^{\mathsf{L}}(p)$; likewise, $\mathsf{name}^{\mathsf{R}} : \mathbf{Int} \rightharpoonup \mathbf{String}$ will have the same meaning in the right domain (hence it is actually the same function as $M^{\mathsf{R}}$).

The consistency relation is defined by $\mathcal{C} \subseteq \mathcal{L}^{\mathsf{L}} \times \mathcal{L}^{\mathsf{R}}$ such that

$$M^{\mathsf{L}}\ \mathcal{C}\ M^{\mathsf{R}} \Leftrightarrow \exists \varphi : \mathrm{dom}(M^{\mathsf{L}}) \leftrightarrow \mathrm{dom}(M^{\mathsf{R}}),\ \mathsf{name}^{\mathsf{L}} = \mathsf{name}^{\mathsf{R}} \circ \varphi$$

In words: a pair of left/right models is related if and only if there is a name-preserving bijection between the person identities in the left and those in the right models. Thus, $\varphi$ is an example of the *evidence* mentioned in the previous section.

This consistency relation exhibits the following properties:

- It is both left-total and right-total;
- It is not left-unique but it is right-unique;
- It implies a unique one-to-one relation on the element level.

**Changes.** In general, a left-change is any triple $(M_1^{\mathsf{L}}, M_2^{\mathsf{L}}, \mu)$ where $M_1^{\mathsf{L}}$ and $M_2^{\mathsf{L}}$ are left models, and $\mu : \mathrm{dom}(M_1^{\mathsf{L}}) \rightharpoonup \mathrm{dom}(M_2^{\mathsf{L}})$ is a partial injective function; A right-change is the same but for right models.

The mapping $\mu$ connects the identities used in the original model to those in the destination model. It is a partial function, meaning that identities can neither be split nor merged by a change, but a person $p$ can be deleted (in which case $p \in \mathrm{dom}(M_1^{\mathsf{L}}) \setminus \mathrm{dom}(\mu)$) or created (in which case $p \in \mathrm{dom}(M_2^{\mathsf{L}}) \setminus \mathrm{cod}(\mu)$). In practice, identities will typically not change at all during updates, except for deletion and creation, and so $\mu$ will always be a partial identity. Even so, it is important to recognise this component; for instance, when identities are reused (a person is deleted from the database and later another person is added, reusing the identity of the first) this does *not* mean it is the same person — and the only way in which this confusion is avoided is through $\mu$. In fact, in a very real sense, the presence of the mapping $\mu$ defines the difference between state-based and delta-based BX approaches.

A left-change is *minimal* if one of the following cases holds:

1. It deletes a single element:
   - $|\mathrm{dom}(M_1^{\mathsf{L}}) \setminus \mathrm{dom}(\mu)| = 1$ and $\mathrm{cod}(\mu) = \mathrm{dom}(M_2^{\mathsf{L}})$
   - $M_2^{\mathsf{L}}(\mu(p)) = M_1^{\mathsf{L}}(p)$ for all $p \in \mathrm{dom}(\mu)$.
2. It creates a single element:
   - $\mathrm{dom}(M_1^{\mathsf{L}}) = \mathrm{dom}(\mu)$ and $|\mathrm{dom}(M_2^{\mathsf{L}}) \setminus \mathrm{cod}(\mu)| = 1$
   - $M_2^{\mathsf{L}}(\mu(p)) = M_1^{\mathsf{L}}(p)$ for all $p \in \mathrm{dom}(\mu)$.
   - $M_2^{\mathsf{L}}(p_i) = (n_i, a_i)$ for $i = 1, 2$, $n_1 = n_2$ implies $p_1 = p_2$.
3. It modifies one of the fields of a single element:
   - $\mathrm{dom}(\mu) = \mathrm{dom}(M_1^{\mathsf{L}})$ and $\mathrm{cod}(\mu) = \mathrm{dom}(M_2^{\mathsf{L}})$
   - For a single $p \in \mathrm{dom}(\mu)$, either
     (a) $\mathsf{name}_1^{\mathsf{L}}(p) \neq \mathsf{name}_2^{\mathsf{L}}(\mu(p))$ or
     (b) $\mathsf{age}_1^{\mathsf{L}}(p) \neq \mathsf{age}_2^{\mathsf{L}}(\mu(p))$ (but not both)
   - $M_1^{\mathsf{L}}(q) = M_2^{\mathsf{L}}(\mu(q))$ for all $q \in \mathrm{dom}(\mu) \setminus \{p\}$.

It can be seen immediately that every change is a composition of a finite sequence of minimal chances. For each of these minimal changes we can formulate update functions that cause them:

1. For all $s \in \mathbf{String}$, $\mathit{delete}(s) \in U^{\mathsf{L}}$ is an update function that deletes the person $p$ with $\mathsf{name}^{\mathsf{L}}(p) = s$.
2. For all $s \in \mathbf{String}$, $\mathit{create}(s) \in U^{\mathsf{L}}$ is an update function that adds a person $p$ with $\mathsf{name}^{\mathsf{L}}(p) = s$.
3. (a) For all $s_1, s_2 \in \mathbf{String}$, $\mathit{setName}(s_1, s_2) \in U^{\mathsf{L}}$ is an update function that changes the name of a person from $s_1$ (in the source model) into $s_2$ (in the target model).
   (b) For all $s \in \mathbf{String}, a \in \mathbf{Int}$, $\mathit{setAge}(s, a) \in U^{\mathsf{L}}$ is an update function that changes the age of a person with name $s$ to $a$.

The right changes and updates are defined analogously, except that, obviously, the age attribute (case 3b) is irrelevant.

**Propagation.** According to our definition of a BX benchmark, we now have to precisely define update propagation for the example. Since we have shown how arbitrary changes can be decomposed into minimal changes, we only have to explain how minimal changes are propagated.[2] Forward propagation is actually already completely defined by demanding correctness as $\mathcal{C}$ is left-unique. We therefore concentrate on backward propagation. Let $M_1^{\mathsf{L}}\ \mathcal{C}_\varphi\ M_1^{\mathsf{R}}$ be a pair of consistent models, and let $\delta^{\mathsf{R}} = (M_1^{\mathsf{R}}, M_2^{\mathsf{R}}, \mu^{\mathsf{R}})$ be a minimal right-change. We define $\delta^{\mathsf{L}} = \overleftarrow{\mathcal{C}}(\delta^{\mathsf{R}}, M_1^{\mathsf{L}})$ for each of the cases of minimal change listed above. Note that age changes do not exist in this setting.

1. For deletion, let $p \in \mathrm{dom}(M_1^{\mathsf{R}}) \setminus \mathrm{dom}(\mu^{\mathsf{R}})$, and define $M_2^{\mathsf{L}}$ as the restriction of $M_1^{\mathsf{L}}$ to all persons except $\varphi^{-1}(p)$. $\mu^{\mathsf{L}}$ is the identity function on $\mathrm{dom}(M_2^{\mathsf{L}})$.
2. For creation, let $p \in \mathrm{dom}(M_2^{\mathsf{R}}) \setminus \mathrm{cod}(\mu^{\mathsf{R}})$ and fresh $q \notin \mathrm{dom}(M_1^{\mathsf{L}})$; define $M_2^{\mathsf{L}} = M_2^{\mathsf{L}} \cup \{(q, (\mathsf{name}_2^{\mathsf{R}}(\mu^{\mathsf{R}}(p)), a))\}$ for some default age $a$. $\mu^{\mathsf{L}}$ is the identity function on $\mathrm{dom}(M_1^{\mathsf{L}})$.
3. Let $p \in \mathrm{dom}(M_1^{\mathsf{R}})$ be the person whose name has changed; let $s_1 = \mathsf{name}_1^{\mathsf{R}}(p)$, $a_1 = \mathsf{age}_1^{\mathsf{R}}(p)$, and $s_2 = \mathsf{name}_2^{\mathsf{R}}(\mu^{\mathsf{R}}(p))$, and let $q = \varphi^{-1}(p)$. Define $\{M_2^{\mathsf{L}} = M_1^{\mathsf{L}} \setminus \{(q, (s_1, a_1))\}\} \cup \{(q, (s_2, a_1))\}$. $\mu^{\mathsf{L}}$ is the identity function on $\mathrm{dom}(M_1^{\mathsf{L}})$.

The interesting cases, for which not all information is available in the right model, are when a person is created (in that case a *default* age has to be inserted) or when a name is changed (in that case the age must be preserved).

**Test cases and BenchmarX repository.** The draft and artifacts related to this first benchmark (to be known as `Person2Person`), and all future BenchmarkXs, will be uploaded to the BX Examples Repository, being set up by Cheney et al. [1] at the BX community Wiki at `http://bx-community.wikidot.com/examples:home`

Several illustrative test cases are already described in the Wiki, tailored for assessing different operational modes of existing BX tools, including the following:

**`Person2Person.BCF`** The goal of this test case is to assess the conformance to BX properties (correctness and completeness) of a forward propagation run in batch mode (i.e, to create a new right-model from an input left-model). This is equivalent to providing as original right-model the empty model with no persons. The inputs for the runs in this test case will be typically small in size, and hand-picked to expose possible corner cases where achieving conformance might not be trivial (in general this will be the case with conformance test cases).

**`Person2Person.BCB`** This test case is similar to the previous, but for the backward propagation.

**`Person2Person.BSF`** The goal of this test case is to assess the performance (scalability of run time and memory consumption) of a forward propagation run in batch mode. A size parametric left-model generator will be provided.

**`Person2Person.BSB`** This test case is similar to the previous, but for the backward propagation.

**`Person2Person.MCB`** The goal of this test case is to assess the conformance to BX properties (correctness, completeness, and hippocraticness) of state based backward propagation. It can be used to assess tools that allow the propagation of changes on the right model to the left one in the style of constraint maintainers (where only the result model of the change is known) [7]. Since the right model contains strictly less information than the left one, the dual test case is equivalent to batch forward propagation.

A summary of the features these test cases intend to assess is presented in Fig. 2.

[2] We thus require for this example that propagation be compatible with update composition, i.e., for readers familiar with the delta-lens framework, we demand PUTPUT [3].

| | Performance | | Scalability | | Laws | | | |
|---|---|---|---|---|---|---|---|---|
| Test Cases | Time | Memory | Time | Memory | Correctness | Completeness | Hippocraticness | Least-change |
| `Person2Person.BCF` | | | | | ✓ | ✓ | | |
| `Person2Person.BCB` | | | | | ✓ | ✓ | | |
| `Person2Person.BSF` | | | ✓ | ✓ | | | | |
| `Person2Person.BSB` | | | ✓ | ✓ | | | | |
| `Person2Person.MCB` | | | | | ✓ | ✓ | ✓ | |

**Figure 2: Feature matrix for the `Person2Person` BX Benchmark**

# 4. RELATED WORK

Benchmarking is an important means of assessing and driving development and improvement in a specific area. In BX related communities, existing benchmarks include for instance [11] for databases, and [12] for graph transformations. These and other existing benchmarks, however, are not directly applicable for BX and cannot be used to address the specific challenges and characteristics of a comparison of BX tools. Our proposal for a BX benchmark format, certainly inspired by existing benchmarks (especially [12]), is explicitly designed to address BX specific aspects.

There also exist tool comparisons and surveys for BX including a quantitative and qualitative comparison of Triple Graph Grammar (TGG) tools by Hildebrandt et. al [5], and the more general survey of BX approaches by Stevens [9]. These survey papers indicate the need for a systematic comparison of existing BX tools but are either too specific (e.g., only covering TGG tools), or too general (no quantitative comparison via a well-defined transformation). Our benchmark proposal is an attempt to fill this gap.

Finally, the Transformation Tool Contest (TTC)[3], organized yearly, is an ideal venue for presenting challenge transformations and soliciting solutions, which are then compared systematically. Although the TTC 2013 actually had a challenge[4] that required bidirectionality, TTC does not typically focus on BX scenarios. Furthermore, the TTC tends to be specialized for model transformation approaches, while BX encompasses other approaches, e.g., from the programming language or database community.

# 5. DISCUSSION AND CONCLUSION

In this paper, we have identified the properties of a BX benchmark and proposed a format for classifying and presenting BX benchmarks. As a proof of concept, we presented the `Persons2Persons` BX benchmark as an example that adheres to our format. Note that our example, however, does not fulfil our "usefulness" criterion as it is meant as a minimal template. The next step is to establish a series of BX benchmarks according to the proposed format, extending and refining it as required. There have already been commitments from BX tool developers including Echo,[5] eMoflon,[6] GRoundTram,[7] and HenshinTGG,[8] to provide, support and implement BenchmarX in the near future.

**Acknowledgements.** The authors would like to thank the reviewers for their insightful comments, and all the participants in the 2013 Banff's meeting on BX that were also involved in the discussion that eventually led to this paper, in particular, Soichiro Hidaka and James Terwilliger.

The second author is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by national funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FATBIT with reference FCOMP-01-0124-FEDER-020532.

[3] `http://planet-sl.org/ttc2013/`
[4] `http://goo.gl/754XT`
[5] `http://haslab.github.io/echo/`
[6] `http://www.emoflon.org`
[7] `http://www.biglab.org/`
[8] `https://github.com/de-tu-berlin-tfs/Henshin-Editor`

# 6. REFERENCES

[1] J. Cheney, J. Gibbons, J. McKinna, and P. Stevens. Towards a Repository of BX Examples. In *Proc. of BX 2014*, 2014.

[2] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective. In R. F. Paige, editor, *Proc. of ICMT 2009*, volume 5563 of *LNCS*, pages 260–283. Springer, 2009.

[3] Z. Diskin, Y. Xiong, and K. Czarnecki. From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. *JOT*, 10:1–25, 2011.

[4] F. Hermann, H. Ehrig, F. Orejas, K. Czarnecki, Z. Diskin, Y. Xiong, S. Gottmann, and T. Engel. Model Synchronization Based on Triple Graph Grammars: Correctness, Completeness and Invertibility. *Software & Systems Modeling*, pages 1–29, 2013.

[5] S. Hildebrandt, L. Lambers, H. Giese, J. Rieke, J. Greenyer, W. Schäfer, M. Lauder, A. Anjorin, and A. Schürr. A Survey of Triple Graph Grammar Tools. In P. Stevens and J. F. Terwilliger, editors, *Proc. of BX 2013*, volume 57 of *ECEASST*. EASST, 2013.

[6] M. Hofmann, B. C. Pierce, and D. Wagner. Symmetric Lenses. In T. Ball and M. Sagiv, editors, *Proc. of POPL 2011*, pages 371–384. ACM, 2011.

[7] L. Meertens. Designing Constraint Maintainers for User Interaction, 1998. Available at `http://www.kestrel.edu/home/people/meertens`.

[8] A. Schürr and F. Klar. 15 Years of Triple Graph Grammars. In H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, editors, *Proc. of ICGT 2008*, volume 5214 of *LNCS*, pages 411–425. Springer, 2008.

[9] P. Stevens. A Landscape of Bidirectional Model Transformations. In R. Lämmel, J. Visser, and J. a. Saraiva, editors, *Proc. of GTTSE 2008*, volume 5235 of *LNCS*, pages 408–424. Springer, 2008.

[10] P. Stevens. Towards an Algebraic Theory of Bidirectional Transformations. In H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, editors, *Proc. of ICGT 2008*, volume 5214 of *LNCS*, pages 1–17. Springer, 2008.

[11] Transaction Processing Performance Council. TPC Benchmark C (Standard Specification, Revision 5.11), 2010. `http://www.tpc.org/tpcc/`.

[12] G. Varró, A. Schürr, and D. Varró. Benchmarking for Graph Transformation. In *Proc. of VL/HCC 2005*, pages 79–88, 2005.