iBench First Cut [Technical Report][*]

Patricia C. Arocena
University of Toronto
prg@cs.toronto.edu

Mariana D'Angelo
University of Toronto
mdangelo@cs.toronto.edu

Boris Glavic
Illinois Institute of Technology
bglavic@iit.edu

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

ABSTRACT

We present iBench, a metadata generator for creating arbitrarily large and complex mappings, schemas and schema constraints. iBench is rooted in the influential STBenchmark system, a benchmark designed for evaluating mapping systems. We extended STBenchmark with several new features, including specialized support for generating logical mappings using the languages of source-to-target (s-t) tuple generating dependencies (tgds) and plain second-order (SO) tgds. iBench can be used with a data generator to efficiently generate realistic data integration scenarios with varying degrees of size and complexity.

1. INTRODUCTION

The study of data integration is as old as the field of data management. However, given the maturity of this area it is surprising that rigorous empirical evaluations of research ideas are so scarce. We argue that a stronger focus on empirical work would benefit the integration community as a whole and identify one major roadblock for this work - the lack of comprehensive benchmarks, scenario generators, and publicly available implementations of quality measures. This makes it difficult to compare integration solutions, understand their generality, and understand their performance for different application scenarios. For a comprehensive survey on benchmarking schema matching and mapping tasks, we refer the reader to Bellahsene et al. [6].

In this technical report, we present iBench[1], a first attempt towards a *metadata* generator for creating arbitrarily large and complex *mappings*, *schemas* and *schema constraints*. iBench extends the well-known STBenchmark [2] system, a benchmark designed for evaluating mapping systems. Our extensions are based on a suite of mapping scenarios commonly used across information integration applications [2] that have been extended to consider various mapping languages, including source-to-target (s-t) tuple generating dependencies (tgds) [8] (also known as Global-and-Local-As-View or GLAV for short [10]) and plain second-order (SO) tgds [4]. iBench can be used with a data generator to efficiently generate realistic data integration scenarios with varying degrees of size and complexity. Our specific contributions are the following.

- Support for generating logical mappings, using the languages of s-t tgds and plain SO tgds, in addition to transformations.
- Support for generating schema constraints, including primary keys (PKs) and random multi-attribute functional dependencies (FDs).
- Support for generating mappings with flexible value invention (also known as Skolemization in the literature).
- Support for new integration scenarios in the form of mapping primitives (e.g., different variants of Vertical Partitioning).
- Sharing of source and target schema elements among multiple instances of mapping primitives.

We envision using iBench to create benchmarks for different integration tasks including (virtual) data integration [10], data exchange [8], schema evolution, mapping operators like composition [9] and inversion [3], and schema matching [12, 13]. The rest of the report is organized as follows. In Section 2, we discuss the architecture of iBench. In Section 3, we describe some noteworthy features of iBench, with an emphasis on the main extensions done to STBenchmark [2]. In Section 4, we briefly report on the first empirical evaluation that has been done using iBench. Last, we conclude in Section 5.

2. IBENCH GENERATOR

Our vision with iBench is to develop a generator that is capable of producing diverse, but realistic inputs and expected outputs (gold standard) for a wide variety of integration tasks. It should also be possible to use the generator to produce micro benchmarks. Ideally, the generator should

[*]Please referred to this document as follows: "iBench First Cut. Technical Report, Department of Computer Science, University of Toronto, 2013.".
[1]iBench stands for **i**ntegration **Bench**mark.

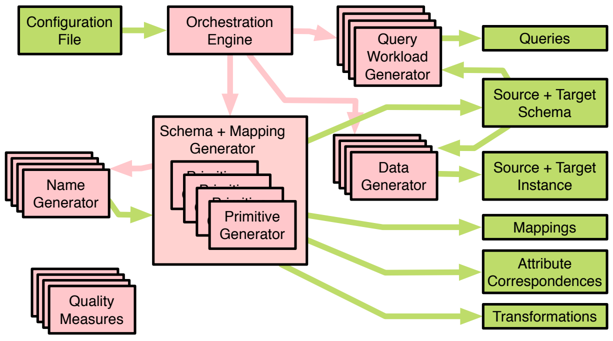

Figure 1: iBench Architecture

include a library of quality functions [1, 11] that are used to compare the output produced by a tool under evaluation with the expected output (if necessary).

Our approach for achieving this goal is as follows. First, we build a generator for simple scenarios that consists of a single source and a single target schema and a mapping between these two schemas. This generator will be able to generate all elements for such a mapping: schemas (including constraints), attribute correspondences, mappings, transformations, instance data, and a query workload. This generator will take a *configuration file* as an input. Scenario elements can be turned on/off individually using the configuration file to generate all inputs and a gold standard for a particular integration task. For example, for schema matching we would only generate the source and target schema (inputs) and attribute correspondences (expected output).

We envision supporting a wide range of integration tasks. In particular, we know that not every integration task may use a single source and a single target schema. For example, schema evolution requires several mapping steps, and in virtual integration, we may have several local schemas that are mapped to the same global schema. We thus propose an *orchestration engine* with iBench that can create complex multi-step and multi-schema integration scenarios by chaining or parallelizing several correlated source-to-target scenarios (as generated by simpler scenario generators). Interestingly, we show that a novel feature, that we call **schema reuse**, needed for a flexible metadata generator also enables the orchestration engine to chain and parallelize the results of simpler scenario generators.

2.1 Metadata Generator

- Concepts of Primitives: Fix a set of typical simple mappings such as vertical partitioning for which we fully understand the semantics and can produce mappings and scenario elements manually. Generalize by customization.

- Generator that takes configuration parameters and creates a complex scenario by combining many primitive instances

- Sharing schema elements between primitives to create more realistic example. This is a main enabler for orchestration too.

- Automatic conversion of real world example into primitives. User provides real-world example fully specified. System wraps this into a new primitive that can then be used in scenario generation.

2.2 Orchestration Engine

The orchestration engine calls the metadata generator several times to create a complex integration scenario. The "shape" of the final integration scenario will be chosen by the user in the configuration file. For example, the user can ask for an integration scenario with three source schemas that are all mapped to a single target schema (usable for virtual integration). There is a general way in which the orchestration engine can achieve parallel mappings (more than one source of target) and sequential mappings (e.g., $S_1 \rightarrow S_2 \rightarrow S_3$).

- $N : 1$: The orchestration engine generates a single $S_1 \rightarrow T$ scenario. Afterwards, the metadata generator is called with 100% target reuse using T to generator $S_2 \rightarrow T, \ldots, S_n \rightarrow T$.

- $1 - 1 - \ldots - 1$: The orchestration engine generate an initial scenario $S_1 \rightarrow S_2$. Afterwards, the orchestration engine calls the metadata generator with 100% source reuse using S_i to produce S_{i+1}.

3. PROTOTYPE IMPLEMENTATION

We have implemented a first prototype of iBench[2], rooted on the influential STBenchmark [2] system. STBenchmark is able to randomly generate a broad range of GLAV mappings by combining basic mapping primitives into complex mappings. The mapping generator takes as input a set of configuration parameters[3] and returns as output a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$. Figure 2 provides a comparison of some noteworthy features in STBenchmark, our iBench (prototype) and the envisioned full-fledge iBench system. Next we discuss how we extended the original STBenchmark in several non-trivial ways.

Our first extension generates, for each mapping scenario, a set of SO tgds instead of executable XQuery mappings. In addition, we extended STBenchmark with new mapping scenarios, including different variants of vertical partitioning primitives, some of which offer specialized support for generating mappings with (possibly repeated) Skolem functions, including functions with both overlapping and disjoint arguments. This permits the modeling of complex correlations, including Skolem functions that cannot be unskolemized and rewritten into GLAV. In Figure 3, we outline the mapping primitives supported by iBench. We use bold face to highlight new mapping scenarios. For each primitive, we offer a brief description and an example mapping, in which we underline those attributes which are part of primary keys, when these are essential for understanding the semantics of the transformation. Moreover, we indicate whether each primitive generates no Skolem terms ($-$), or some Skolem terms, using fixed or variable Skolemization strategies (*Fixed*

[2]Available at `www.dblab.cs.toronto.edu/project/iBench` and as a git repository at `https://bitbucket.org/bglavic/ibench/`.

[3]We refer the reader Alexe et al. [2] for a detailed discussion of these parameters.

Element Type	STBenchmark	iBench (prototype)	iBench
Only Schemas	No	Yes	Yes
Name Generation	dictionary	dictionary	dictionary + configuration
Schema Constraints	No	PK, FK, arbitrary FDs	PK, FK, arbitrary FDs
Attribute Correspondences	No	Yes	Yes
Logical Mappings	No	s-t tgds & plain SO tgds	s-t tgds & plain SO tgds
Source Instance	XML (using ToXGene)	relational (CSV)	XML/relational (plugins)
Target Instance	by running transformations	by running transformations	by running transformations/direct
Transformations	SQL with XQuery	SQL	SQL (exchange/virtual) / XQuery
Query Workload	No	No	Yes
Errors	No	No	Yes

Figure 2: Scenario Element Type Support

Name	Description	Example	Skolems
ADD	Copy a relation and add new attributes	$R(a,b) \rightarrow S(a,b,f(a,b))$	Variable
ADL	Copy a relation, add and delete attributes in tandem	$R(a,b) \rightarrow S(a,f(a))$	Variable
CP	Copy a relation	$R(a,b) \rightarrow S(a,b)$	-
DL	Copy a relation and delete attributes	$R(a,b) \rightarrow S(a)$	-
HP	Horizontally partition a relation into multiple relations	$R(a,b) \land a = c_1 \rightarrow S_1(b)$ $R(a,b) \land a = c_2 \rightarrow S_2(b)$	-
ME	Inverse of vertical partitioning (merge)	$R(a,b) \land S(b,c) \rightarrow T(a,b,c)$	-
MA	Inverse of vertical partitioning + adding attributes	$R(a,b) \land S(b,c) \rightarrow T(a,b,c,f(a,b,c))$	Variable
OF	Object fusion, e.g., inverse of horizontal partitioning	$R(\underline{a},b) \rightarrow T(a,b,f(a))$ $S(\underline{a},c) \rightarrow T(a,g(a),c)$ $R(\underline{a},b) \land S(\underline{a},c) \rightarrow T(a,b,c)$	Fixed
SJ	Copy relation (S) and create a relationship table (T) through a self-join	$R(\underline{a},b,c) \rightarrow S(a,c)$ $R(\underline{a},b,c) \land R(\underline{b},d,e) \rightarrow T(a,b)$	-
SU	Copy a relation and create a surrogate key	$R(a,b) \rightarrow S(\underline{f(a,b)},b,g(b))$	Fixed / Variable
VH	Vertical partitioning into a HAS-A relationship	$R(a,b) \rightarrow S(\underline{f(a)},a) \land T(\underline{g(a,b)},b,f(a))$	Fixed
VI	Vertical partitioning into an IS-A relationship	$R(\underline{a},b,c) \rightarrow S(\underline{a},b) \land T(\underline{a},c)$	-
VNM	Vertical partitioning into an N-to-M relationship	$R(a,b) \rightarrow S_1(\underline{f(a)},a) \land M(\underline{f(a),g(b)}) \land S_2(\underline{g(b)},b)$	Fixed
VP	Vertical partitioning	$R(a,b) \rightarrow S_1(\underline{f(a,b)},a) \land S_2(\underline{f(a,b)},b)$	Variable

Figure 3: Mapping Primitives

and *Variable*, respectively). Note that the original STBenchmark only uses Skolem functions in surrogate key (SU) and vertical partitioning (VP) scenarios, and only a single function in each. Note as well that the example mappings represent the simplest version of each primitive. The generated mappings may be much more involved and diverse. For instance, a vertical partitioning may split a relation into more than two fragments (depending on how the generator configuration parameters are set). For primitives that support variable Skolemizations, we support three different strategies (i.e., *Key*, *All*, and *Random*). Which strategies are applied is determined using a configuration parameter called *Skolem Mode*.

We also enhanced STBenchmark to support the generation of primary keys and FDs over the source. New configuration parameters, *Primary Key FDs* and *Source FDs*, are used to turn these features on and off as desired. We support the random generation of source FDs, including the generation of multi-attribute and partial FDs. We also added support for reuse of schema elements across primitives. Thus, when creating a schema mapping scenario, our modified version of STBenchmark may decide to reuse previously created source and target relations. For example, two instances of a *CP* primitive may copy from the same source relation. This is of importance for generating correlations among mappings and in a sense, more realistic cases of Skolem terms across clauses used in complex SO tgds.

Composition of schema mappings, like schema evolution, can lead to complex interactions between Skolem terms (specifically the arguments of the Skolem functions). In addition

Parameter	Min	Max
Number of Primitives (per type)	0	10
Number of Attributes Per Relation	2	15
Number of Key Attributes	1	3
Join Path Length	2	4
Join Type	Star	Chain
Primary Key FDs	No	Yes
Source FDs	0%	50%
Skolem Noise	0%	50%
Source Reuse	0%	50%

Figure 4: Some Configuration Parameters

to Skolem terms introduced in our primitives, we introduce additional Skolem terms by randomly choosing source attributes to be replaced (in the target) with Skolem functions. We use a new configuration parameter, called *Skolem Noise*, to indicate the fraction of source attributes that should be assigned Skolem terms. Whenever a variable v is used in an atom at the position corresponding to one of the "Skolemized" attributes, we replace all occurrences of this variable in target atoms with the Skolem function with randomly generated arguments based on the remaining source variables. For instance, assume the Skolem term $f(a)$ was assigned to attribute b of relation $S(a,b)$. We would transform the SO tgd $S(x_1,x_2) \rightarrow T(x_1,x_2)$ into $S(x_1,x_2) \rightarrow T(x_1,f(x_1))$.

4. APPLICATION

We briefly report on the first empirical evaluation that has been done using iBench [5]. We used iBench to evaluate when schema mappings with large amounts of incompleteness have a first-order (FO) semantics. This involved evaluating a number of rewriting algorithms over a large set of realistic mapping scenarios, in which the degree and complexity of incompleteness (i.e., the Skolem functions used for modeling value invention) could be controlled. We used iBench to generate schemas, schema constraints (including keys and functional dependencies), and schema mappings expressed as SO tgds. We also implemented a generator of random configuration files which was input to iBench to produce several million integration scenarios of varying size and complexity. Figure 4 outlines some important configuration parameters involved in the generation of random configurations. For example, we generated configurations where we randomly selected the number of instances for each mapping primitive (from 0 to a maximum of 10, following a uniform distribution). The size of the source and target schema was determined not only by the number of requested attributes per relation (plus/minus deviations), but also by the type of requested primitives and some other additional parameters, such as the length of join paths. We consider variable types of joins (i.e., star and chain), key sizes, percentages of source FDs, Skolemization strategies, Skolem noise, and source reuse.

5. CONCLUSION

Developing a system like iBench is an ambitious goal. We presented (and have released) a first prototype implementation of iBench and discussed how it was an essential tool in a large scale empirical evaluation we have conducted in previous work [5]. Much remains to be done, for example, given the focus on accuracy in integration research and the work on debugging and repairing integration solutions [7], we also plan to extend iBench to systematically create metadata with different types of errors so that repair solutions can be evaluated.

Acknowledgments. We thank Bodgan Alexe, Wang-Chiew Tan and Yannis Velegrakis for making the source code implementation of STBenchmark available to us. Arocena was supported by the NSERC Business Intelligence Network. D'Angelo was supported by the NSERC Undergraduate Student Research Award (held during the Summer of 2012).

6. REFERENCES

[1] B. Alexe, M. A. Hernández, L. Popa, and W. C. Tan. MapMerge: Correlating Independent Schema Mappings. *VLDB J.*, 21(2):191–211, 2012.

[2] B. Alexe, W. C. Tan, and Y. Velegrakis. STBenchmark: Towards a Benchmark for Mapping Systems. *PVLDB*, 1(1):230–244, 2008.

[3] M. Arenas, J. Pérez, J. Reutter, and C. Riveros. Inverting Schema Mappings: Bridging the Gap between Theory and Practice. *PVLDB*, 2(1):1018–1029, 2009.

[4] M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. The Language of Plain SO-tgds: Composition, Inversion and Structural Properties. *J. Comput. Syst. Sci.*, 79(6):763–784, 2013.

[5] P. C. Arocena, B. Glavic, and R. J. Miller. Value Invention in Data Exchange. In *SIGMOD Conference*, pages 157–168, 2013.

[6] Z. Bellahsene, A. Bonifati, F. Duchateau, and Y. Velegrakis. On evaluating schema matching and mapping. In *Schema matching and mapping*, pages 253–291. Springer, 2011.

[7] L. Chiticariu and W. C. Tan. Debugging Schema Mappings with Routes. In *VLDB*, pages 79–90, 2006.

[8] R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[9] R. Fagin, P. Kolaitis, L. Popa, and W. C. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. *TODS*, 30(4):994–1055, 2005.

[10] M. Lenzerini. Data Integration: a Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[11] G. Mecca, P. Papotti, S. Raunich, and D. Santoro. What is the IQ of your Data Transformation System? In *CIKM*, pages 872–881, 2012.

[12] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB J.*, 10(4), 2001.

[13] P. Shvaiko and J. Euzenat. A Survey of Schema-Based Matching Approaches. *J. Data Semantics IV*, pages 146–171, 2005.