Well Constructed Workflows in Bioinformatics

Anna Gambin[1], Jan Hidders[2], Natalia Kwasnikowska[3], Sławomir Lasota,[1]
Jacek Sroka[1], Jerzy Tyszkiewicz[1], Jan Van den Bussche[3]
[1]Warsaw University,
[2] University of Antwerp
[3] University of Limburg

October 15, 2004

Abstract

We demonstrate and discuss the advantages of bioinformatical workflows which have no side-effects over those which do have them. In particular, we describe a method to give a formal, mathematical semantics to the side-effect-free workflows defined in SCUFL, the workflow definition language of Taverna. This is achieved by translating them into a natural extension of the Nested Relational Calculus NRC.

1 Introduction

Bioinformaticians conduct their research by reasoning about large amounts of data. For each experiment they use specialized software tools, either by installing their local copies or by accessing them via Internet. These systems are then organized into a kind of network through which the data flows and is processed. The flow is often organized in an *ad hoc* manner, either by manual invoking of the necessary tools, or by shell scripts. Such a system can be called a *workflow*, by an analogy to similar systems found in the sciences or industry. But what really counts in bioinformatics is the data manipulation and not mere invoking of particular operations. So bioinformatical workflows are typically *dataflows*.

There are special software systems for defining and executing bioinformatical workflows. One of them is *Taverna* [6]. At present time the workflow definition language of Taverna, called *SCUFL*, allows for a range of abstraction levels [1]. The user can apply Taverna for web service orchestration and define his own bioinformatical operations as workflows. Such operation definitions are usually complicated because of the need to communicate with stateful web services (i.e., ones which simulate over HTTP a session with the user, by means of a session id), which can even provide messages (e.g., that the computation will take more time), to which the workflow must properly react (e.g., to avoid accepting such a message as the result of the computation). These operation definitions can be then nested, to yield more complicated bioinformatical workflows.

ShowGOIntersection (see Fig. 1) workflow example, distributed with Taverna, is a good example of a workflow whose goal is to define a session with a stateful web service, i.e., a single operation from the datacentric point of view. A quick glance at this example reveals that it does not represent a real dataflow. It's output is obtained thanks to side-effects of using certain functions of a stateful web service. The only data item really flowing through this workflow is the session ID which in a call to the corresponding web service is transformed into the result. The other operations consume their input and move the data to the web service's memory. An extensive synchronization mechanism has to be used to ensure the correct order of operations.

It is our opinion that this level of abstraction is not really appropriate for bioinformatics. A workflow constructed from many basic operations whose real functionalities are achieved by using side-effects is not a well-constructed one—note that in procedural programming languages, writing procedures with side-effects is generally regarded as a bad programming style. It is much better to keep the necessary impurities of the programming method local in separated procedures and functions, whose interactions are then already purely functional.

We suggest the same method for workflows: a well constructed bioinformatical workflow is one in which there are no side-effects. Of course, certain web services are stateful, and therefore cannot be used as such in dataflows. A simple solution is to wrap them, so that they appear as black-box functions without side-effects to the workflow. Our focus in this paper is on the side-effects-free workflows, and we do not intend to look inside their black-boxes. Such workflows (which we call *dataflows* below) have a number of advantages over those which permit side-effects.

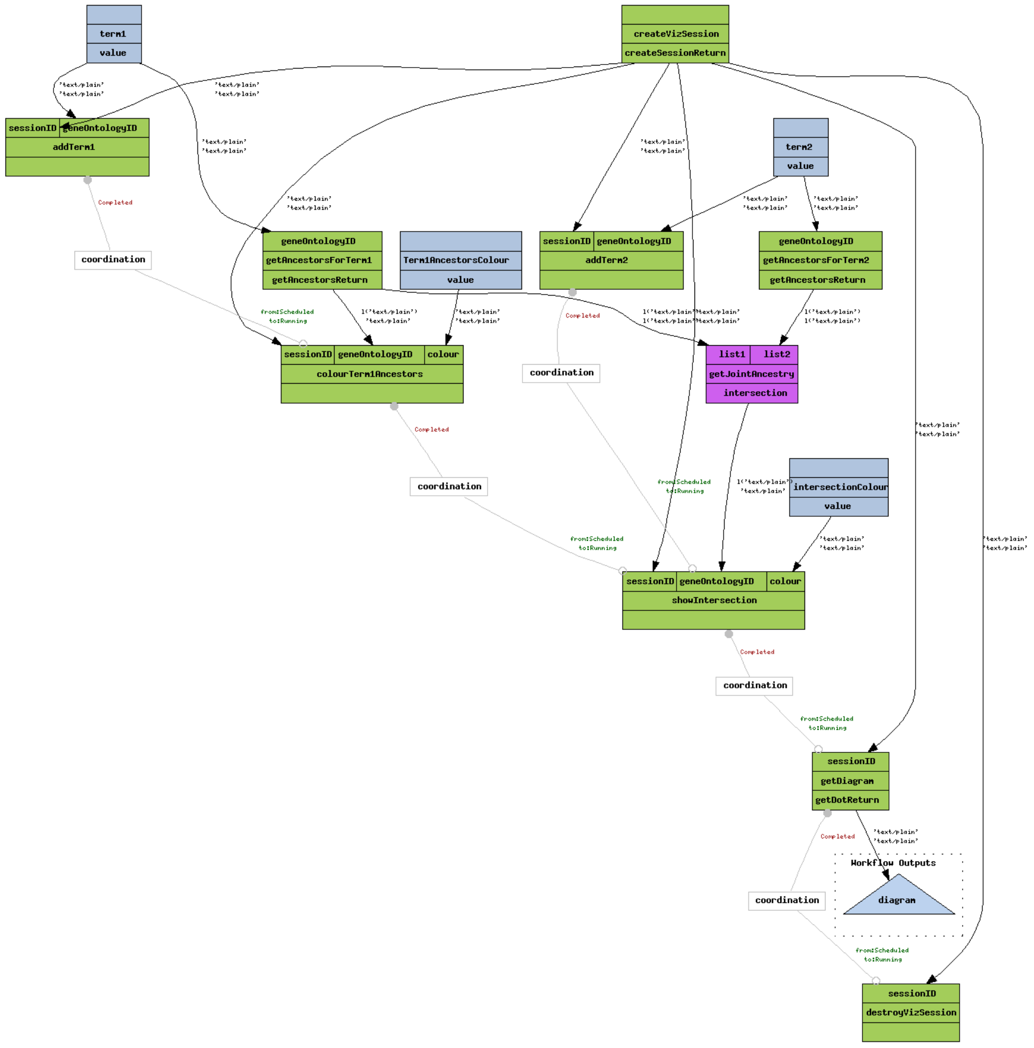

Figure 1: ShowGOIntersection workflow example from Taverna

- It is our opinion that formal, and yet executable, descriptions of all *in silico* experiments of bioinformatics should be published along with their biological conclusions. Used for that purpose, dataflows can be much more easily analysed and understood, which is crucial for:
  - Debugging by the authors.
  - Effective and objective assessment of their merit by the publication reviewers.
  - Easy understanding by the readers, once published.
- Improvement and modification of dataflows is much easier, as the individual operations can be substituted by other ones of the same functionality. This allows also for independent verifications of their results.
- Dataflows are independent of the changes in the web service functionality (even though the interfaces to the web services need to be changed), which makes them executable for a longer period of time.

In this paper we demonstrate results, which provide several other arguments in favor of dataflows.

- Dataflows admit a translation into a natural extension of the *nested relational calculus (NRC)*, which is a standard theoretical model of complex objects developed in the database community. This yields:

  - A formal, mathematical semantics for dataflows.
  - A theoretical framework in which one can reason about dataflows, simplify and optimize them.

- Dataflows can be meaningfully treated as data items themselves, and as such stored in repositories along with their outputs and queried. This opens the following possibilities:
  - Searching for already existing dataflows of analogous functionality, thus preventing one from redoing already existing research, and giving the reviewers a chance to identify unoriginal work.
  - Searching for already existing dataflows which realize fragments of a planned experiment, which saves time and resources.
  - Automated creating and invoking of dataflows as the results of queries applied to dataflows found in the repository (e.g., "execute a dataflow A with inputs provided by all the dataflows B in the repository which did BLAST and ever yielded a given sequence X among their top 10% of the results, sorted by the score").

The above effects cannot be reasonably achieved for workflows with side-effects. We do not speak here about a theoretical impossibility, however. We believe that there is an urgent need of *dataflow engineering*, the counterpart of *software engineering*, for dataflows of bioinformatics. We think it is extremely important to create good principles of constructing dataflows, which can be used by every bioinformatician to conduct good, correct and efficient experiments *in silico*. And we think from the workflow engineering point of view it is counterproductive to attempt to achieve the above goals for workflows with side-effects. It is clear that the cost of maintaining software grows at least linearly with the size of its code. From this point of view, drawing 15 boxes and dozens of edges to do each action on the data (have a look at the picture above) means increasing the complexity of checking the correctness of the workflow by a factor of at least 15. Of course, the total size of the code does not decrease just by wrapping web services, but these can be then maintained by professional software engineers, and not by bioinformaticians. Next, the once a web service access is wrapped and tested for correctness, it can be used over and over again and does not really count in the maintenance costs any more. This is a classical advantage of separating software libraries from the application code.

To feel that difficulty, the reader is asked to follow ShowGOIntersection and make sure that its goal is really achieved: the service is properly activated, and the output is eventually computed.

Related research. The (unextended) NRC has been already used as the core language to express a kind of workflows in the BioKleisli system [3]. However, BioKleisli's focus was only on specifying and running the workflows, while our intentions go in the direction of dataflow engineering and are therefore much broader. The ISXL project [8] has motivations similar to ours. However, it introduces a completely new ad-hoc language, while NRC was already existing and ready to be used formalism. Again, from the engineering point of view it is always better to stick to an existing and well understood solution than to introduce a new one, unless the latter is intended to remedy important shortcoming of the former one.

2 Technical matters

Informally, we consider a workflow to be well-constructed (and call it a dataflow) if the inputs to all its basic operations are transferred in an explicit way (i.e., there is no external memory, no state of an external web service, etc.), and the only effect of executing an external function is its output (i.e., there are no side-effects). Consequently, in dataflows there is no need of synchronization or functions which yield no output. There is no unique formal definition of a workflow. In fact, there are many such competing definitions, one for each of the systems, like BioKleisli, Taverna, ISXL, etc. Therefore we present our ideas on a particular example. Our choice is to give a formal semantics for the dataflow part of SCUFL, the workflow definition language of Taverna. We define the semantics indirectly, by translating dataflows into an extension of the Nested Relational Calculus (NRC) [2]. This extension will be denoted eNRC in the sequel.

Nested relational calculus (NRC) is a core language allowing to describe functional programs using collection types, e.g. lists, bags, sets, etc. A most important feature of the language is the possibility to iterate over a collection. The only collections used in the following are lists, hence in the description below we ignore other collection types of NRC.

NRC contains a set of base types. Moreover, it is allowed to combine these types to form records and lists. NRC is strongly typed, but we do not focus on this aspect here.

Besides standard constructs enabling manipulation of records and lists, NRC contains three constructs **sng**, **map** and **flatten**. For a value v of a certain type, **sng**(v) yields a singleton list containing v. Operation **map**, applied to a function from type τ to σ, yields a function from lists of type τ to lists of type σ. We use the following notation for **map**$(f)(X)$:

$$[f(x)|x \in X].$$

Finally, the operation **flatten**, given a list of lists of type τ, yields a flattened list of type τ, by concatenation. These three basic operations are powerful enough for specifying functions by structural recursion over collections, cf. [2].

The inspiration for these constructs comes from the category-theoretical notion of a *monad* and the monadic approach to uniformly describe different notions of computation [5]. Similarly as in computational lambda calculus [5], each nontrivial computation processing a collection type yields always a collection type again. Even if there is a single final value, it will be returned as a singleton list.

We extend NRC by the following two additional constructs:

- **zip** on lists defined semantically by **zip**$([a_1, \ldots, a_n], [b_1, \ldots, b_m]) = [\langle a_1, b_1\rangle, \ldots, \langle a_{min(n,m)}, b_{min(n,m)}\rangle]$. Consequently, **zip** outputs an empty list if either of its arguments is empty.

  zip is necessary to model the *dot product* of Taverna and cannot be simulated in pure NRC.

- External functions

  $$f : \tau \to \sigma.$$

  External functions are intended to model the behavior of the properly wrapped web services, treated in a dataflow as black-boxes. Such functions have been already considered in the context of pure NRC [4].

It is also possible to add internal function definitions, by allowing one to give names to well formed expressions of eNRC with one input, and use them henceforth in the dataflow, under the restriction that no form of recursion is permitted. This is however only syntactic sugar.

It is worth noting that the definition of SCUFL it is permitted to use recursive definitions of internal functions, which really increases expressive power of the language. However, we haven't seen any example which makes use of this possibility and, crucially, do not see any really meaningful way to terminate such a recursion (which could happen only if eventually the body of the recursive loop got an input but produced no output—which is forbidden in dataflows).

3 SCUFL vs. eNRC

It would be tempting to formulate a theorem stating that *for every side-effects-free workflow expressed in SCUFL there is an equivalent expression in eNRC*. However, this is impossible because there is no formal semantics of SCUFL. Therefore our translation of the dataflows of SCUFL into eNRC should be regarded as a method to *provide a formal semantics for the side-effect-free fragment of SCUFL*.

Due to the space limitations we cannot give the translation in the present paper, deferring it to the full version. Instead, as a matter of example, we encode the (badly constructed) example from the figure.

$$[[\Pi_1(\langle \texttt{getDiagram}(sid), \texttt{showIntersection}(sid, join, \texttt{intersectionColour}())\rangle) \\ \mid sid \in \textit{flatten}(\textit{flatten}(shortcut))] \mid sid \in [\texttt{createVizSession}()]]$$

where $shortcut$ should be substituted by

$$[[\Pi_1(\langle \texttt{getJoinAncestry}(l1, l2), \texttt{colourTerm1Ancestors}(sid, l1, \texttt{Term1AncestorsColour}())\rangle) \\ \mid l1 \in \textit{flatten}(shortcut1)] \mid l2 \in \textit{flatten}(shortcut2)]$$

where again $shortcut1$ should be substituted by

$$[\Pi_1(\langle \texttt{getAncestorsForTerm1}(t1), \texttt{addTerm1}(sid, t1)\rangle) \mid t1 \in [\texttt{term1}()]]$$

and $shortcut2$ should be substituted by

$$[\Pi_1(\langle \texttt{getAncestorsForTerm2}(t2), \texttt{addTerm2}(sid, t2)\rangle) \mid t2 \in [\texttt{term2}()]].$$

However, this large expression is semantically (in the sense of eNRC) equivalent to nothing more than

$$\texttt{getDiagram}(\texttt{createVizSession}())$$

because our tricks to represent an operation which yields no output ($\Pi_1(\langle u, v\rangle)$) and of synchronization ($[u|x \in [[v]]]$, where x is not a free variable in u) are semantically vacuous.

This perverse example shows that we can indeed model workflows in eNRC, and that one can think of (manual or automated) optimization of dataflows. In this case, the underlying assumption of side-effect-freeness leads to a conclusion that the workflow does nothing. Similarly, even in a well-constructed ones, one can aim at detection and removal of semantically vacuous fragments of workflows. Indeed, the authors of BioKleisli [3] have already considered optimization and found it to be practically doable. It is not surprising, since eNRC allows one to specify certain data manipulations typical for database querying—an area where automated optimization is an industrial standard.

4 Further research

Our topics for further research are as follows:

- A graphical language for eNRC, to have a "graphical syntax" for the side-effect-free fragment of Taverna. Taverna's original graphical language is not very well suited for this, as part of its semantics is determined at runtime. E.g., if an external function does not accept a whole list as its input, it starts iterating over it.
- A query language for dataflows, able to select and manipulate their structure to yield new dataflows. We hope the experience of querying the syntax and outputs of SQL queries in Meta-SQL [9] can be useful here.
- Optimization techniques for dataflows. There are many ways to define the cost of an execution of a dataflow, and equally many optimization problems.

References

[1] M. Addis, J. Ferris, M. Greenwood, P. Li, D. Marvin, T. Oinn, A. Wipat Experiences with e-Science workflow specification and enactment in bioinformatics In *Proc. UK e-Science All Hands Meeting*, pages 459–467, 2003.

[2] P. Buneman, S. Naqvi, V. Tannen and L. Wong Principles o programming with complex objects and collection types. *Theoretical Computer Science*, 149:3–48, 1995.

[3] S. Davidson, C. Overton, V. Tannen and L. Wong BioKleisli: A Digital Library for Biomedical Researchers *Journal of Digital Libraries*, 1(1):36-53, Nov 1997.

[4] L. Libkin, L. Wong Conservativity of Nested Relational Calculi with Internal Generic Functions. *Inf. Process. Lett.* 49(6): 273-280(1994).

[5] E. Moggi Notions of computation and monads, *Information and Computation* **93** (1991) pp 55-92

[6] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows *Bioinformatics*, 2004. Accepted for publication.

[7] R. Stevens, H.J. Tipney, C. Wroe, T. Oinn, M. Senger, C.A. Goble, P. Lord, A. Brass, and M. Tassabehji. Exploring Williams-Beuren Syndrome using $^{\mathrm{my}}$Grid. In *Proceedings of 12th International Conference on Intelligent Systems in Molecular Biology*, 2004. Accepted for Publication.

[8] A. Troger, A.A.A. Fernandes A language for comprehensively supporting the in vitro experimental process in silico *Proc. 4th IEEE International Symposium on BioInformatics and BioEngineering (BIBE 2004)*, pages 47- 56, March 2004.

[9] J. Van den Bussche, S. Vansummeren, G. Vossen Meta-SQL: Towards practical meta-querying . System demonstration paper in *Advances in Database Technology, EDBT 2004*, Lecture Notes in Computer Science, vol 2992, pages 823-825. Springer, 2004.