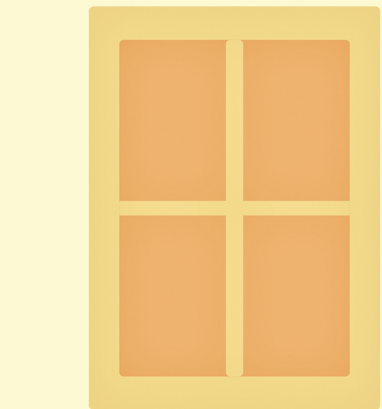
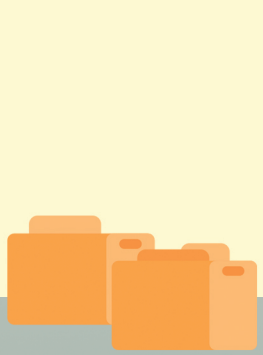
Synthesis of Scientific Workflows:
Theory and Practice of an Anstance-Aware Approach





Vedran Kasalica

Synthesis of Scientific Workflows: Theory and Practice of an Instance-Aware Approach

Synthese van Wetenschappelijke Workflows: Theorie en Praktijk van een Instantiebewuste Benadering

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Utrecht University op gezag van de rector magnificus, prof.dr. H.R.B.M. Kummeling, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op maandag 21 november 2022 des middags te 12.15 uur.

door

Vedran Kasalica

geboren op 2 April 1992
te Kotor, Montenegro

Promotor:
Prof.dr. G. Keller

Copromotor:
Prof.dr. A.-L. Lamprecht

Za ljeta u Komarnici

Contents

CHAPTER 1

Introduction

Contemporary science across all disciplines is increasingly computational, and many scientists regularly face the need of producing software themselves to become able to solve their specific data analysis problems. Many of these programs are essentially *computational pipelines*, i.e., sequences of calls to existing computational components, where the new program is mainly responsible for the coordination of the flow of data between them [11].

Scientific workflow management systems (WMS) support researchers in assembling computational components into complex scientific workflows [2, 12, 16, 95, 147]. WMSs facilitate workflow execution and monitoring directly within the same framework. However, they typically require the users to know (1) which tools are well (or best) suited for the task, (2) how to connect them to solve the specific problems, (3) which connections are possible with regard to the compatibility of input and output data types and formats, and other kinds of technicalities.

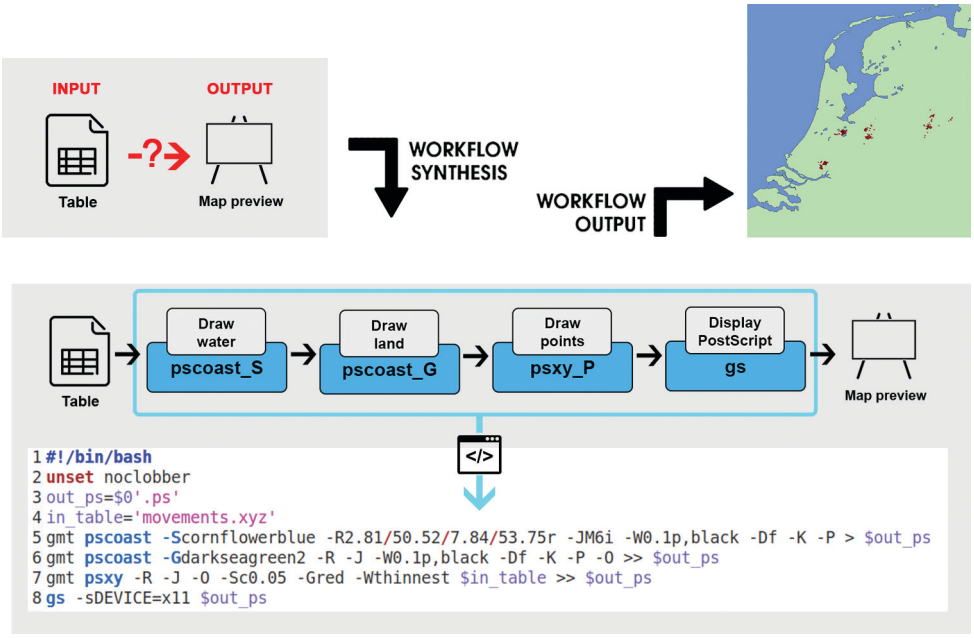


Figure 1.1: An example of a workflow composition (synthesis) process.

Semantics-based automated workflow composition techniques strive to assist users in the discovery and composition of purpose-specific workflows [26, 81, 90, 105]. Ideally, users would only need to state their intentions about the workflow at an abstract, conceptual level (e.g. by providing information about the available inputs and intended output data, or particular kinds of operations to use or avoid), and the workflow environment would automatically translate the specification into a con-

crete executable workflow (as illustrated in Figure 1.1). This is essentially a case of *program synthesis* (considered to be one of the central problems in the theory of programming [113]), which, in the general formulation, aims to find a program that meets a given specification.

Let us consider a geovisualisation scenario where the user, e.g., a biologist, wants to plot bird migration data to assess the relation between bird migrations and the land topography. In order to do so, the user has to specify the scientific workflow that would process the topography and migration data and visualise it on a map. However, the user could, instead of manually selecting each workflow step, specify the goal as a synthesis problem and use the process to compose the desired solution automatically. A synthesis process is illustrated in Figure 1.1. The user specifies the available data and the desired output, and uses the synthesis process to automatically compose a workflow solution. The solution is then executed to get the intended map as a result.

A similar approach could be implemented in the domain of life sciences or any other scientific field that uses computational components. Converting experimental biological data into interpretable results increasingly involves the combination of multiple, diverse computational tools into pipelines or workflows performing specific sequences of operations [11, 44, 134]. Figuring out which tools are applicable (and in which order), and scientifically meaningful is often hard in practice, particularly when the tools were not developed by the same research group, consortium or company. The idea of automated workflow synthesis is to let an algorithm perform or assist the user in this process.

Workflow synthesis approaches rely heavily on well-defined and rich domain annotations. However, such annotations are hard to find for an arbitrary scientific workflow. That is why many of the current approaches that aim to automate workflow composition restrict the problem and focus on either well-defined and curated domains [145] or the automation of individual workflows steps, instead of the workflow as a whole [46, 87].

The international eScience community has created a comprehensive infrastructure of tools, services and platforms that support the work with scientific workflows. Notable results are (1) the EDAM ontology [64] of bioscientific terms, which provides semantical annotations of the domain terminology, (2) bio.tools [66], a publicly curated tools annotation repository in bioinformatics and the life sciences, and (3) the Core Concept Data (CCD) ontology of geo-analytical terms [120]. Availability of these semantic annotations of various domains allows for advancement in practical usage of program synthesis methods.

In this dissertation, I focus on temporal logic-based approaches to program synthesis, and in particular on a new synthesis approach based on SLTL^x (Extended Semantic Linear Time Logic) [72].

The approach is based on SLTL (Semantic Linear Time Logic) synthesis, originally proposed by Steffen et al. [130]. The SLTL-based approach has been used for the automated composition of scientific workflows in the PROPHETS [108] framework (within the jABC working environment [131]) and demonstrated to be useful in various studies [3, 89, 111]. Inspired by the SLTL-based formalism and based on the lessons learned by its application to scientific workflows, this dissertation aims

to improve the approach and provide a workflow synthesiser tailored to scientific applications. In the early phase of my research, two observations were central, (1) the SLTL-based loose programming approach is unable to distinguish data instances, and (2) the close integration with the jABC framework made it difficult to connect PROPHETS to the software ecosystem of the eScience community. First, scientific workflows frequently reuse already generated data in later stages, and might accumulate multiple different data instances with the same type signature. These have to be distinguishable and separately identifiable to ensure correct data transfer between the components of the workflow. However, in SLTL models, states are collections of available type propositions. As a result, when there are multiple data instances of the same type, their signatures are identical and the framework is not able to distinguish them. This leads to ambiguity in the interpretation of the synthesised solutions, which may prevent the creation of executable workflows. Second, to facilitate uptake by practitioners, the new implementation should aim to simplify the import of semantic domain knowledge and export of synthesised workflows in formats that are commonly used in the eScience community. In addition, it should provide its functionalities through an API, a GUI and potentially a CLI.

1.1 Methods

This section presents the methodology I used while conducting the research presented. My goal is to provide a practical solution to a scientific question, and that comes with some specific challenges common to computer engineering tasks.

The difference between solving a problem in theory and in practice can be quite substantial. To provide an approach that can solve synthesis problems in practice one needs to be aware of the questions the scientists might pose, as well as of the knowledge that is currently available in various domains, i.e., its content and the formats in which it can be presented.

My goal is to provide a framework, grounded in a sound theory, which is able to successfully solve problems which occur in practice. This is a methodology commonly used in Software Engineering, which motivated me to adopt the core principles of Agile (Research) methodology [14, 135] in my research. I interpret the core principles as follows.

1. **Individuals and Interactions over Processes and Tools** - In our case, the needs of scientists in various domains should be the priority. The infrastructure and existing domains are important, but the needs of individuals should drive the development of our formalism.
2. **Working Software over Comprehensive Documentation** - In our case the theory and implementations are not required to be documented in detail (in form of a paper) after each improvement, but rather aim at making sure that the formalism captures the desired features.
3. **Customer Collaboration over Contract Negotiation** - The goal is to collaborate with users (scientists) to improve our formalism, as opposed to developing a framework individually and expecting users (scientists) to adapt their infrastructure accordingly.
4. **Responding to Change over Following a Plan** - Considering that I did not

know all the desired features of such a formalism up front, I had to be flexible and change our priorities over time.

My research comprises four recurring steps (1) discovering desired features, (2) formally defining the features (and extending the existing theory to facilitate them, when needed), (3) implementing the formalism and (4) evaluating the formalism on real examples (as illustrated Figure 1.2). This entails that this dissertation, apart from the theoretical and practical contributions, presents a number of case studies I performed in collaboration with specialists from various scientific domains. The studies aim to evaluate the introduced formalism and motivate its improvements.

Figure 1.2: Agile Research Methodology

1.2 Contributions

My research, as presented in Figure 1.2, revolves around a practical solution (the APE framework) to the workflow synthesis process. Over the years I have researched various scenarios that required workflow synthesis. They both, motivated and evaluated my approach. Figure 1.2 presents four key cycles that contributed to the final framework. Each of the cycles is characterised by a specific APE release, its main feature and a case study(es) that motivated its improvements. This section identifies my three main contributions in the process.

SLTL^x-specified Transducer Synthesis The main theoretical contribution of this dissertation is the introduction of the Extended Semantic Linear Time Logic (SLTL^x). I developed it in collaboration with Natasha Alechina and Brian Logan [72]

and it is presented in Chapter 3. The logic formalises workflow properties and the approach employs a Transducer Synthesis to synthesise the corresponding solutions.

SLTL^x is, simply put, an extension of Linear Time Logic (LTL) that uses labelled edges, first-order elements and term taxonomies to enrich the semantics. An SLTL^x model can be interpreted as a workflow that satisfies a given SLTL^x formula (temporal goal). The model comprises a set of multi-transducers[1] depicting workflow operations and a corresponding transducer-port binding. The port binding represents a dependency between operations, where operation inputs depend on existing operation outputs. The individual instances (input and output transducer-ports) are crucial to fully automate workflow composition and to keep the information about the data provenance.

In addition, I introduce the concept of data dimensions, disjoint sets of properties where each set characterises a specific aspect (dimension) of data. In an n-dimensional domain, a data type is characterised by an n-tuple of dimensional properties. For example, in the life sciences domain each data object is characterised by a data type and a data format (e.g., *Mass spectrum* in *Thermo RAW* format), therefore it is a 2-dimensional domain. This concept further enriches the domain annotations, simplifies the problem specification and improves the quality of the synthesis results.

Automated Pipeline Explorer The second major contribution presented in this dissertation is a framework that implements the introduced formalism. Chapter 4 introduces a mechanism to translate the workflow synthesis problem for a given SLTL^x encoding into a propositional format [75]. Such encoding allows the usage of an off-the-shelf SAT (Boolean satisfiability problem) solver, such as MiniSAT [38], to compute the candidate solutions. Chapter 5 follows with the implementation of the formalism in a tool - the APE (Automated Pipeline Explorer) framework [74]. I implemented APE as a library that supports a CLI (command line interface), API (application programming interface) and GUI (graphical user interface).

Case studies in Geo- and Life-sciences The third major contribution is a set of case studies done in collaboration with scientists from geo- and life-sciences, as presented in Chapter 6. This goes beyond the traditional computer science field. The main goal of the case studies is to guide the further development of formalism. My work on setting up the domain annotations and encoding a use case in the geovisualisation domain [73] and the collaboration with geoscientists on question-answering in GIS [84, 120] helped to better understand the importance of distinguishing data instances on the underlying logic level (e.g., by introducing SLTL^x). In addition, the GIS case studies motivated the concept of data dimensions. Originally I assumed that an arbitrary domain would not characterise data with more than two disjoint properties (data type and format). However, GIS case studies demonstrated that our formalism must support an arbitrary number of data dimensions, as some semantic domain annotations require up to four dimensions to characterise data.

The second goal of the case studies is to demonstrate the applicability and to evaluate the quality of the approach. While each case study contributes toward this goal, the collaborative work on workflow synthesis in the proteomics domain [76] stands out. It demonstrates the potential of the SLTL^x workflow synthesis approach

[1] A multi-transducer is a finite deterministic automaton with multiple input and output ports (i.e., it takes a tuple of k inputs and produces a tuple of l outputs, k, l > 0).

"in the wild", as it uses APE over a publicly available domain, without any further annotations, to solve given problems. Furthermore, the case study guides the future annotation processes by pointing out important improvement points and expected challenges.

The dissertation is structured as follows. Chapter 2 presents the background of the workflow synthesis problem as well as the SLTL-based synthesis formalism we build on. Chapter 3 introduces formally the SLTL^x-based synthesis and presents the complexity of three classes of SLTL^x-based synthesis problems. Chapter 4 presents the transformation formalism, used to translate a given domain and problem specification into propositional logic, with the goal of using SAT solving techniques to reason over it. Chapter 5 introduces the APE (Automated Pipeline Explorer) framework, which implements the aforementioned synthesis approach. Chapter 6 presents the case studies I performed in collaboration with scientists from life- and geo-sciences to assess the usability of APE v2 (the latest version of APE). Chapter 7 presents the runtime evaluation of the APE v2 framework and observations obtained from a survey of the APE framework users. Chapter 8 concludes the dissertation with a discussion of some related applications and potential future directions for the framework.

Note that, in the remaining chapters I use the academic "we" when I refer to any of my contributions, as most of my research was performed in collaboration with external researchers.

CHAPTER 2

Background

Abstract - The dissertation introduces a novel scientific workflow synthesis, i.e., automated composition, approach, which relies on an existing temporal (SLTL)-based approach and builds upon it. This chapter provides an overview of the existing program synthesis approaches, and focuses on workflow synthesis as their subset. Unlike the program synthesis approaches, which often have to limit the structure of synthesised programs to provide efficient automated composition approaches, scientific workflow synthesis has, by definition, well-defined and structured composition goals. The scientific workflows can be represented as acyclic directed graphs, and are used as such in practice. This loop-free structure allows for an efficient synthesis, without compromising the workflow structure. The temporal logic (SLTL)-based synthesis approach presented in the chapter, allows for an automated composition of scientific workflows based on an abstract description of the problem. The approach is used as a basis for the contributions presented in the following chapters.

The scientific workflows are some of the main structures used to process data in modern computational science, as presented in the introductory chapter. Chapter 1 describes the difficulties that contemporary scientists are facing due to the increased amount of computational tools available. Distinguishing and identifying tools that are needed for a task is increasingly hard. Therefore, semantics-based automated workow composition, i.e., workflow synthesis, techniques have been used to simplify the process. This chapter presents the background of the workflow synthesis research presented in the dissertation. Section 2.1 provides an overview of the program synthesis as a whole, before Section 2.2 presents scientific workflow synthesis problems. Finally, Section 2.3 presents in detail the temporal logic-based workflow synthesis approach, which inspired and supported the research presented in the dissertation.

2.1 Program Synthesis

Unlike typical compilers that translate well-defined high-level languages to machine code using sets of syntactic rules, program synthesis is typically accomplished by performing some type of search over the search space of programs that are consistent w.r.t. a specification, usually resulting in more than one possible solution. Two major challenges in program synthesis are the state explosion of the search space [97, 137], caused by the combinatorial nature of the problem, and the correct interpretation of the user intent. Both are non-trivial problems that were tackled from different angles throughout the years. This resulted in the development of various synthesis techniques [20, 53], as well as their application to many different domains. According to Gulwani [50], each synthesis approach can be characterised by three essential dimensions: (1) the format in which the *user intent*, i.e., the problem specification, is provided, (2) the *search space* of candidate programs in which it searches, and (3) the *algorithm* used to perform the search.

User intent is an obvious choice for a key characteristic of synthesis approaches. It defines the interaction between the user and the synthesis framework, essential for the applicability of the approach. The main goals, when modelling a user intent, are to provide an intuitive technique for describing the problem specification (what is considered intuitive typically depends on the targeted users) and to remove ambiguities in the specification.

Early synthesis approaches relied on the existence of a complete and formal specification of the program. Some approaches used theorem provers to construct a proof of the user specification, and the logical program itself [48, 102], while others used program transformations over abstract program specifications to produce the desired low-level programs [101]. However, providing the initial specification proved to be as complex as writing the program itself. This led to a focus shift [125, 126, 132] from deductive program specifications to inductive specifications, such as input-output examples, partial specification, etc. It has become common practice to have an interactive loop between the user and the synthesis algorithm, where the user, based on the provided candidate solutions in the previous step, can provide additional examples or specification constraints to resolve ambiguities.

Many current approaches use *formal logic descriptions* to provide the user intent [128]. The description is used to capture the logical relation between the program input and the program output. However, these types of specifications are usually hard to construct, as they require the user to be familiar with the underlying logic. Although the work presented in this dissertation falls under this category, similarly to the PROPHETS framework for loose programming [91, 108], it uses natural-language templates for providing the specification, and so accounts for users not trained in logics or formal specification. This type of user intent would, according to [50], be categorised between logical and natural language specifications.

Although the formal specification allows for an accurate description of the user intent, end users might not find it intuitive and straightforward. To solve this issue, some approaches focus on an *example-based specification* format [49, 51] to model the user intent. This type of problem specification allows users to provide examples of desired outputs based on given inputs.

Some approaches allow for a *partial description* of the program as part of the specification of the user intent. These approaches are also called *sketching*, implemented by the Sketch system [127]. Loose programming, as it is used in PROPHETS, is another example of the same underlying idea. As an example from the eScience community, the WINGS (Workflow Instance Generation and Selection) framework [46] employs the idea of providing a workflow template, where individual steps can be left out and are automatically included based on the context when the template is instantiated.

Finally, programmers might consider a programming language as the best tool for specifying their intent. This is applied in superoptimisation of code [52, 112], and in the synthesis of program inverses [35], such as compression/decompression, encryption/decryption, etc.

The search space is defined by the structures that can be provided as a synthesis output, as well as by the restrictions made on the problem implementation. Furthermore, its size and complexity are crucial for the computational complexity of the synthesis problem. The search space should keep a balance between the expressive power of the framework and the efficiency of a search over it. In other words, it should be comprehensive enough to support a large set of candidate programs, and at the same time restrictive enough to support efficient search mechanisms. Synthesis approaches tend to limit the search space in some way to improve their runtime performance.

In practice, the search space can vary from programs in general programming languages to such in domain-specific formalisms. It is defined by the supported *operators* and *control structures*. The approach presented in this dissertation targets programs that restrict control structure to linear/sequential programs, also referred to as *loop-free programs*. Another such approach is the previously mentioned superoptimisation approach [52], implemented using SMT solvers. Loop-free programs can express a wide range of computations, such as text-editing programs [93, 106], API call sequences [100], and unbounded data type manipulations [88].

Other approaches allow the user to provide a skeleton (grammar) of the space of possible programs in addition to the specification [7]. As the grammar provides a

structure for the hypothesis space, these approaches can yield more efficient search procedures. Additionally, a strict grammar ensures better interpretability of the candidate solutions. Examples of such approaches include the Sketch [127] and WINGS systems, and the looping templates described by Srivastava et al. [128].

The search technique can be based on enumeration search algorithms, deduction, constraint solving, statistical techniques, or a combination of them. The approach presented in this dissertation uses constraint solving techniques, also categorised as *logical reasoning-based techniques*. The main idea is to reduce the synthesis problem to a SAT problem, and then use an off-the-shelf SAT solver to explore the search space. The reduction typically involves two steps: *constraint generation* and *constraint solving*. The constraint generation procedure involves the generation of the logical constraints, such as the logical relations between inputs and outputs, whereas the resolving of the constraints yields the desired program. The latter involves the translation of the generated logical constraints into the corresponding SAT constraints and the usage of the SAT solver as the synthesis reasoner. Counterexample-guided inductive synthesis (CEGIS) is another popular solving technique. It originates from Counterexample-guided abstraction refinement (CEGAR) [28] in combination with debugging using counterexamples [124]. CEGIS is an inductive synthesis approach where synthesis is driven by counterexamples usually provided by a constraint solver. Examples of systems that utilise CEGIS are PYCO [61], a tool that performs constrained synthesis from component libraries and the aforementioned Sketch system.

2.2 Scientific Workflow Synthesis

This dissertation focuses on computational pipelines, commonly referred to as scientific workflows. These loop-free programs can be represented as finite acyclic directed graphs, where nodes depict operations, i.e., computational tools, and edges depict data and/or control flow dependencies. As mentioned in the introductory chapter, scientific workflows play a key role in modern computational science research, such as life science [92] and geo-science [6]. Data analyses must be tailored to highly complex data and processes, hence, scientists regularly use sophisticated workflows, composed of several software tools and data resources.

The problem of *scientific workflow synthesis*, given a computational problem specification, is to produce a scientific workflow that satisfies the specification. This simplification of the problem, when compared to the general program synthesis, allows researchers to optimise the synthesis approach and provide more lightweight solutions. For example, systems, such as Wings [46] and the tool recommender system in Galaxy [87] focus on finding individual tools to fill in the gaps in existing workflows. These types of approaches synthesise a workflow by instantiating the missing steps within the provided workflow structure. On the other hand, systems such as AI planning in GIS [39, 150] Magallanes [116], SHARE [138] and HYDRA [13, 115] and PROPHETS [108] synthesise complete workflows based on abstract specifications. These approaches rely on well-annotated domains that allow them to automatically chain together compatible operations and compose valid workflows. The Magal-

lanes system, as well as the AI planning in GIS approaches, focus on composing web services in bioinformatics and geo-science, respectively. The SHARE and HYDRA query engines allow the generation of fully executable pipelines. To accomplish that, they rely on well-annotated and curated semantic domain annotations - SADI registry [144], as well as the SPARQL query language. The PROPHETS framework provides a synthesis of scientific workflows based on temporal logic specifications. It requires semantic annotations that describe the domain vocabulary and operation with respect to their inputs-output dependencies. Finally, synthesis approaches, such as the one provided by Nextflow [33], focus on synthesising optimal workflow executables. Instead of synthesising the structure of the workflow, they utilise a given data-flow structure to implement optimisations, such as the parallelism, in the resulting executable file.

This dissertation focuses on the synthesis of complete workflows as they provide a broad field of application. More concretely, it focuses on the temporal logic-based synthesis approach behind the PROPHETS system, as it supports workflow synthesis in an arbitrary semantically annotated domain. The approach has an additional advantage, that unlike some of the other approaches, it can work with limited domain annotations. Such semantic annotations are readily available in some scientific domains, such as the life-sciences [92, 111].

PROPHETS is a plugin to the jABC modelling framework for eXtreme model-driven development (XMDD) [104, 131]. It allows workflow developers to mark connections between workflow building blocks as “loosely specified” and run the synthesiser to turn the loose specification into a fully specified and executable workflow part. The specification is provided in the Semantic Linear Time Logic (SLTL) [130], which is an extension of the Linear Time Logic (LTL). Users can formulate additional constraints for the loose specification that the synthesiser takes into account. Therefore PROPHETS provides a constraint editor with natural-language constraint templates, which the users can easily fill with terms from a domain-specific controlled vocabulary.

The following section describes the SLTL-based synthesis approach which underlines the PROPHETS framework. The dissertation uses the approach as a base that it builds upon.

2.3 SLTL-based Workflow Synthesis

The SLTL-based synthesis method was initially proposed by Steffen et al. [41, 130], following the revised formulation in [91]. The section presents the syntax and semantics of SLTL, as well as the SLTL-based workflow synthesis problem.

Workflow synthesis with the SLTL-based method relies on semantic annotations, i.e., a domain model, about the data types and operations in the targeted application area. *Domain models* comprise data type/operation taxonomies (referred to as Tax_D/Tax_O) as controlled vocabularies, and a set FA of semantic annotations of the available tools using terms from the domain taxonomies.

Definition 1. *A taxonomy is a weakly connected directed acyclic graph $G = (V, E)$ where the vertices V are terms in θ, which describes entities of a domain, and the directed edges E define relations between the entities. Taxonomies have a designated*

root element $v_0 \in V$ that has no outgoing edges. All other elements in V have at least one outgoing edge.

The set of all abstract (D_a) and concrete types (D_a) in the domain model is denoted by $D = D_a \cup D_c$, and the set of all abstract (O_a) and concrete operations (O_c) by $O = O_a \cup O_c$. The type taxonomy is a taxonomy with $\theta = D$ and the operation taxonomy is a taxonomy with $\theta = O$. Terms from these taxonomies are used for the semantic tool annotation in the domain.

Definition 2. *A semantic tool annotation is a triple (o, Use_o, Gen_o)[1], where $o \in O_c$ is a concrete operation (tool) from the domain, $Use_o \subseteq D$ is the set of types that must be available before its execution (i.e. its input types) and $Gen_o \subseteq D$ is the set of types that are created by its execution (i.e. its output types). The set FA comprises all semantic tool annotations of the domain model.*

The semantic tool annotations FA define the *synthesis universe*, which constitutes the search space in which the synthesis algorithm looks for solutions to the synthesis problem. It combines the domain knowledge into an abstract representation of all possible solutions.

Definition 3. *The synthesis universe is a triple $(2^D, O_c, Trans)$ where*

- *D is a set of concrete and abstract data types.*
- *O_c is a set of concrete operations (tools).*
- *$Trans = (d, o, d')$ is a set of transitions where d, $d' \in 2^D$ and $o \in O_c$.*

The synthesis universe can be constructed from the semantic tool annotations as follows: For each $d \in 2^D$, a state in the universe is created. The transition (d, o, d') is added to $Trans$ iff $Use_o \subseteq d$ and $d' = d \cup Gen_o$. Each path in the synthesis universe represents a possible workflow. Note that albeit potentially very large, the synthesis universe is finite. It can however contain loops and therefore represent infinite paths (workflows). The *synthesis problem* is to find (finite) paths p in the synthesis universe that satisfy the formal specification Φ of the intended scientific workflow ($p \vDash \Phi$), provided in the form of an SLTL formula.

Definition 4. *For a given taxonomy over a set of terms θ, taxonomy expressions are defined as follows:*

$$TE ::= a \mid \neg TE \mid TE \wedge TE \mid TE \vee TE$$

where $a \in \theta$ is a term from the taxonomy.

Definition 5. *Semantic Linear Time Logic (SLTL) is a semantically enriched version of linear time logic (LTL) that is focused on finite paths. The syntax of SLTL is given by the following BNF:*

$$\Phi ::= true \mid type(d_c) \mid \neg \Phi \mid \Phi \wedge \Phi \mid < o_c > \Phi \mid \boldsymbol{G}\,\Phi \mid \Phi\, \boldsymbol{U}\, \Phi$$

where d_c and o_c represent taxonomy expressions over types and operations, respectively.

Definition 6. *Let $(2^D, O_c, Trans)$ be the synthesis universe and p an alternating sequence of type sets and operations, defined as $p = (d_0, o_1, d_1, o_2, d_2, ..., d_{k-1}, o_k, d_k)$*

[1] The original definition also includes the sets $Kill$ (defines those types that are destroyed and therefore removed from the set of types that were available prior to execution of the operation) and $Forbid$ (a set of types that must not be available before execution of the operation), but we omit them here as they are not commonly used, nor relevant for the workflow synthesis problems that we address with this dissertation.

where the workflow bound $k \in \mathbb{N}_0$, $d_i \in 2^D$ and $o_i \in O_c$. Path p (or p_0) satisfies formula Φ ($p_0 \vDash \Phi$) in SLTL, under $(2^D, O_c, Trans)$, according to the following definition:

$$
\begin{array}{lll}
p_i \vDash true & & \textit{true for every path} \\
p_i \vDash d & \textit{iff} & DTax(d_i) \vdash d \textit{ (under propositional logic)} \\
p_i \vDash \neg\Phi & \textit{iff} & p \nvDash \Phi \\
p_i \vDash \Phi_1 \wedge \Phi_2 & \textit{iff} & p \vDash \Phi_1 \wedge p \vDash \Phi_2 \\
p_i \vDash < o > \Phi & \textit{iff} & k > i \textit{ and } OTax(o_{i+1}) \vdash o \textit{ (under} \\
 & & \textit{propositional logic) and } p_{i+1} \vDash \Phi \\
p_i \vDash \boldsymbol{G}\Phi & \textit{iff} & \forall x \in \{i, ..., k\} : p_x \vDash \Phi \\
p_i \vDash \Phi_1 \boldsymbol{U} \Phi_2 & \textit{iff} & \exists x \in \{i, ..., k\} : \forall y \in \{i, ..., x-1\} : \\
 & & p_y \vDash \Phi_1 \textit{ and } p_x \vDash \Phi_2
\end{array}
$$

where p_i is defined as:

$$
\begin{array}{lll}
p_i = (d_i, o_{i+1}, d_{i+1}, ..., o_k, d_k) & & when \quad i \in \{0, ..., k-1\} \\
p_i = (d_k) & & when \quad i = k
\end{array}
$$

while the used functions for the evaluation of the taxonomic information, $OTax : O_c \to 2^O$ and $DTax : 2^{D_c} \to 2^D$ are defined as follows:

$$
\begin{aligned}
&OTax : x \mapsto \{o \mid o \in drv_O(x)\} \\
&DTax : X \mapsto \{d \mid \exists x \in X : d \in drv_D(x)\}
\end{aligned}
$$

with the taxonomy is-a relation utilised to create a set of derivable terms for each $a \in \theta$, by the following recursive definition:

$$drv_\theta(a) = \{a\} \cup \{X | \exists a' \in \textit{is-a}(a, a') : X \in drv_\theta(a')\}$$

In addition to the *globally* ($\boldsymbol{G}$) and *until* ($\boldsymbol{U}$) operators as defined above, we will use two additional operators to simplify the notation: $\boldsymbol{X}\Phi$, interpreted as $< true > \Phi$, denotes the *next-time* operator, and $\boldsymbol{F}\Phi$, interpreted as $true\,\boldsymbol{U}\,\Phi$, denotes *finally* operator.

The SLTL-based synthesis problem is defined as follows.

Definition 7 (SLTL workflow synthesis)**.** *The **SLTL workflow synthesis problem** is: given a synthesis universe $(2^D, O_c, Trans)$ and an SLTL formula Φ (the goal formula), is there an alternating sequence of type sets and operations p that satisfies the formula Φ ($p \vDash \Phi$) under $(2^D, O_c, Trans)$.*

The PROPHETS framework implements a tableau algorithm that solves SLTL workflow synthesis problems, having exponential worst-case complexity [130]. The complexity, as well as the fact that the library is used within a larger jABC framework, reflects on the runtime. For example, PROPHETS exceeds a timeout of one hour when synthesising workflows of length 10 in the geovisualisation domain [73]. The framework provides an additional implementation, based on a monadic second-order logic[2]. The runtime does not show drastic improvements in the runtime when

[2]Monadic second-order logic is a second order logic where no function variables are allowed and the relation variables are required to be monadic, i.e., of arity one [56].

synthesising larger workflows. The synthesis approach is, however, not publicly available and thus is not assessed further.

The SLTL-based synthesis approach allows for an exhaustive exploration of the synthesis universe, covering and evaluating all possible operation combinations. However, as discussed in the following chapter, a major limitation of this method in practice is its inability to distinguish data instances. This is due to the semantics of the underlying temporal logic, and the representation of states as sets of data types.

To be able to reason over different instances of the same kind of data, and to provide the user with a formalism that allows full control over the solutions, the approach has to be improved. In addition, the development of jABC, the PROPHETS' working environment, has been discontinued and superseded by the work on the Cinco SCCE Meta-Tooling Suite [109, 110]. To continue this work on automated workflow composition a replacement for the PROPHETS framework is needed, as the framework is closely integrated into the jABC ecosystem. The following chapters present the extension of the SLTL formalism, as well as a new implementation of the formalism, that addresses the mentioned limitations.

CHAPTER 3

From SLTL to SLTL^x

Abstract - A major limitation of temporal logic-based approaches to automatically synthesising a workflow to accomplish a particular computational task from a set of computational tools, is their inability to distinguish data objects with the same type signature. This leads to ambiguity in the specification of the required solution, which may prevent the creation of an executable workflow. This chapter introduces a workflow synthesis approach that is able to keep track of data objects. We view synthesis as a problem of orchestrating transducers representing computational tools to achieve a temporal logic specification. We show that the bounded SLTL^x workflow synthesis problem (where the maximum number of times each tool is used is known in advance) is NP-complete, and the dynamic SLTL^x workflow synthesis problem (where the number of times a tool is used is not known in advance) is PSPACE-complete. Finally, this chapter shows how the SLTL^x-based approach overcomes the limitations of previous approaches, using case studies from the GIS domain for illustration.

This chapter is based on the following publication:

Kasalica, V., Alechina, N., Lamprecht, A.-L. & Logan, B., "Instance-Aware Synthesis of Workflows Specified in Temporal Logic", *Journal of Artificial Intelligence Research (JAIR)*, 2023, Submitted and under review.

The creation of scientific workflows can be challenging. Workflow developers need to identify the relevant workflow components from often large collections of computational tools, and compose them correctly (order, type compatibility) to solve a given computational problem. Automating workflow synthesis reduces the required time and the likelihood of errors.

This chapter introduces a scientific workflow synthesis approach that is able to keep track of data objects. The chapter is structured as follows. Section 3.1 presents the limitations of the SLTL-based synthesis approach, which we try to overcome. Section 3.2 presents semantically annotated multi-transducers as the formal background. In Section 3.3 we describe an extension of SLTL with first-order features, and our new approach to transducer orchestration with temporal goals in Section 3.4. Section 3.5 presents the application to the aforementioned geovisualization case study and evaluates the new approach with respect to the existing SLTL-based approach. Section 3.6 presents related approaches which tackle the synthesis of computational tools.

3.1 Challenges in SLTL Workflow Synthesis

The SLTL (Semantic Linear Temporal Logic) [130] synthesis approach has been used for the automated composition of scientific workflows in the PROPHETS [108] and APE v1.0 [74] frameworks. SLTL is an extension of Linear Temporal Logic (LTL) which introduces labelled edges and term taxonomies to enrich the semantics. An SLTL model can be interpreted as a workflow which satisfies a given SLTL formula (temporal goal). The model contains states representing sets of available data types, and edges representing the operations performed over the data.

The original SLTL-based synthesis approaches suffer from two main limitations: an *inability to distinguish data objects of the same type* and an *inability to express data-tool dependencies*. This leads to ambiguity in the specification of the required solution, which may prevent the creation of an executable workflow. To solve that problem, we extend Semantic Linear Temporal Logic (SLTL) to be able to talk about objects (data instances). We call the resulting logic SLTL^x. Under this formalism, we view synthesis as a problem of orchestrating transducers representing computational tools to achieve a temporal logic (SLTL^x) specification.

Distinguishing data objects is crucial, as scientific workflows frequently reuse data generated in earlier stages, resulting in multiple different data objects with the same type signature. These objects must be distinguished to ensure correct data transfer between the components of the workflow. However, in SLTL models, states are sets of available type propositions. As a result, when there are multiple data objects of the same type, their signatures are identical. This leads to ambiguity in the interpretation of a required solution (which operation should be applied to which data object), and may prevent the creation of an executable workflow.

As an example, consider the synthesis of geovisualisation workflows for generating maps depicting bird movement patterns in the Netherlands [73], as briefly mentioned in Chapter 1. The synthesis goal is to generate a workflow over an existing set of GIS tools that can be used to plot bird movement data and city coordinates

(both provided as CSV files) on a map. The map should plot the water and land, while the bird movement and city coordinates should be plotted as lines and points, respectively. The workflow specification, therefore, indicates the two CSV files (i.e., of type "CSV") containing coordinates of cities and bird movements as inputs. The specification further requires that operations "*Plot water*", "*Plot coast*", "*Plot points*" and "*Plot lines*" are used, and that an output of type "*PostScript*" is produced. The corresponding specification in SLTL (where $\boldsymbol{F}$ means "eventually", $\langle Plot\_water \rangle \Phi$ means Φ holds after applying operator $Plot\_water$, and $\boldsymbol{X}$ means "in the next state", the full syntax is presented in Chapter 2) is given by:

$$CSV \wedge CSV \wedge \boldsymbol{F}(\langle Plot\_water \rangle \boldsymbol{F}(\langle Plot\_coast \rangle (\boldsymbol{F} \\ (\langle Plot\_points \rangle \boldsymbol{F} \langle Plot\ lines \rangle true))) \wedge \boldsymbol{F}(PostScript \wedge \neg \boldsymbol{X} true)$$

However, as the two inputs have the same type ("CSV"), the SLTL specification cannot express that the cities should be depicted as points, whereas the bird movements should be connected with lines, or even that both input files should be used. Figure 3.1 shows some possible interpretations of two different SLTL models satisfying the specification generated by APE v1 [74]. The rectangles represent operations performed, ellipses represent data objects used, and the arrows depict data flows. The arrows also indicate if the data are an input for the operation (red, dotted) or an output of the operation (green, solid). Figures 3.1(a) and (b) correspond to two different interpretations of the shortest model (with respect to the number of operations performed). However, although the model satisfies the SLTL specification, neither interpretation uses both of the inputs: interpretation (a) does not use the bird movement data, while (b) does not use the city coordinates. Figures 3.1(c) and (d) are interpretations of a "longer" model, which performs two transformation operations, instead of one. Interpretation (c) is indeed a valid solution to the problem. The workflow creates a simple map of the Netherlands, depicting the sea as blue, the coast as green and the bird movements as dots on the map. In contrast, interpretation (d) does not use the bird movement data.

To encode such dependencies, existing approaches typically rely on workarounds. For example, our previous work [73] proposes an incremental approach where each segment of the workflow (not containing multiple objects of a data type) is synthesised separately and then composed to produce the final map annotations; e.g., once the city locations are plotted, the corresponding workflow is extended by a newly synthesised workflow that plots the bird movement. However, such workarounds assume that the target workflow can be decomposed into independent sub-problems (which is not always the case), and do not enable full automation of workflow synthesis.

The inability to distinguish data objects also means it is not possible to **express data-tool dependencies**, i.e., to specify properties that relate tools to existing data. This can result in synthesised workflows containing *redundant operations*, that is, the same type of operation being performed over the same data multiple times. For many solutions of length n, the solver can create a solution of length $n + 1$ that performs the same type of operations and introduces a redundant operation that is

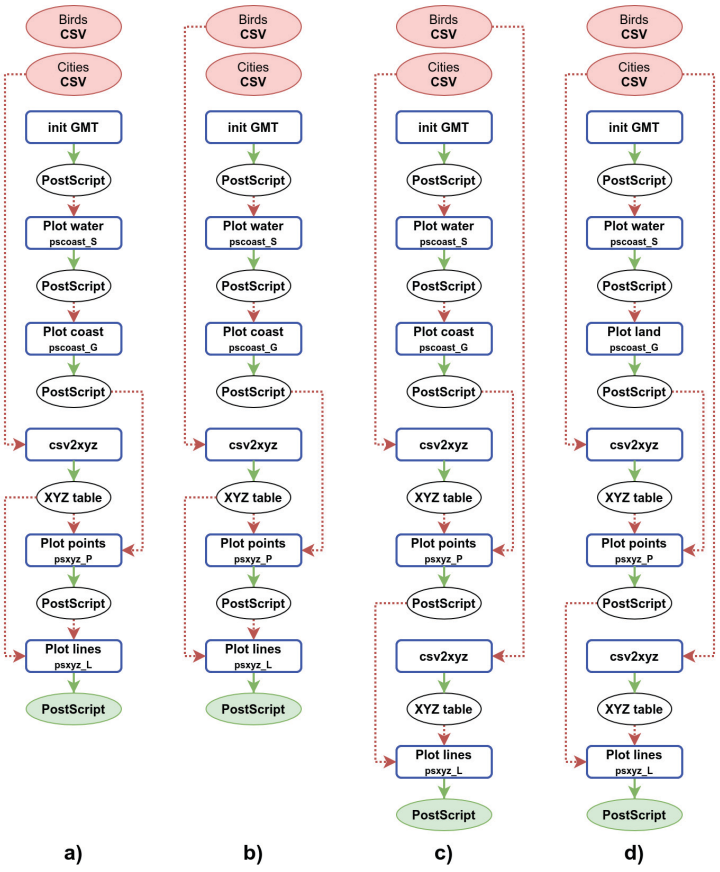
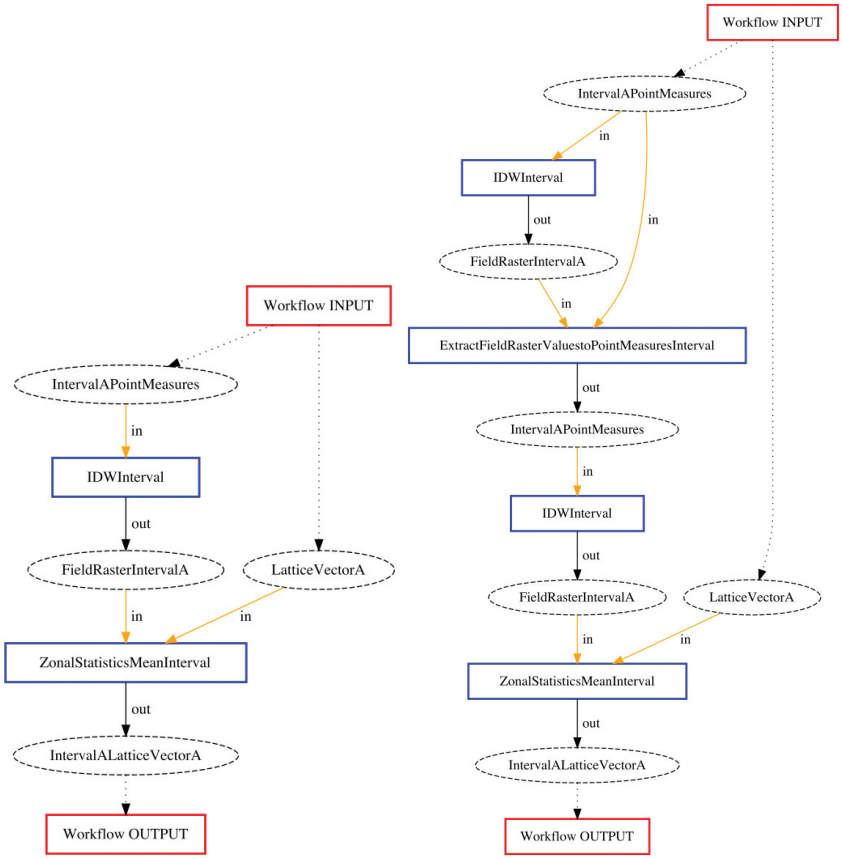


Figure 3.1: Possible synthesis solution interpretations.

consistent with the workflow specification, for example, repeating the same transformation of data multiple times. To illustrate, [84] evaluated the 20 shortest solutions for 10 different workflow synthesis scenarios in geoinformation systems (GIS) and found that 70% of the workflows generated by APE v1 contained such redundant operations (Figure 3.2 illustrates one of such errors). Figure 3.2a presents a correct workflow that was synthesised for the given question. The workflow comprises two operations (blue rectangles), and thus, is considered to be of length 2. In contrast, Figure 3.2b presents a workflow of length 4 that contains redundancy errors. The main redundancy comes from the usage of the *IDWInterval* (marked in red) transformation tool multiple times over the same object (first workflow input).

In this chapter, we present a new approach to the synthesis of workflows, which preserves and utilises information about data objects in a workflow. Our approach combines and extends workflow synthesis using the temporal logic SLTL and controller synthesis for transducers [5, 32] originally developed for the automated generation of controllers for manufacturing facilities. Referring to individual objects



(a) Correct, with no errors.

(b) Correct, with a redundancy error.

Figure 3.2: Example of a redundancy error in workflows synthesised for question "What is the average temperature within each PC4 area in Amsterdam?"

(physical and virtual) is essential in the manufacturing domain. However, there are significant differences in our approach; for example, in the specification formalism and in the assumption that workflows are acyclic. We show that the bounded SLTL^x workflow synthesis problem (where the maximum number of times each tool is used is known in advance) is NP-complete, and the dynamic SLTL^x workflow synthesis problem (where the number of times a tool is used is not known in advance) is PSPACE-complete.

The following chapters aim to solve the bounded synthesis problem in practice. The idea is to translate the SLTL^x specification into propositional logic and use the MiniSAT [38] solver to synthesise the solutions. The translation into propositional logic [75] is presented in Chapter 4, while the APE (for Automated Pipeline Explorer) [74] framework that implements the approach is presented in Chapter 5. The APE v2 is the latest version of our framework that implements the SLTL^x-based

approach and extends on the previous APE v1 that captures the SLTL-based approach.

3.2 Transducers

We model the tools used in workflow synthesis as multi-transducers as in [32]. A transducer is a finite deterministic automaton with outputs [59]. A multi-transducer has multiple input and output ports (i.e., it takes a tuple of k inputs and produces a tuple of l outputs, $k, l > 0$). However, unlike [32], we add semantic annotations on the transitions of a transducer to constrain the types of symbols that can be used as inputs, and also specify the types of outputs. We assume that the annotations come from some set of unary predicates L_T and that each input and output for each transition is annotated with zero or finitely many predicates from this set. Transitions of a transducer with k input and l output ports correspond to $k + l$-ary relations from the set of predicates L_O. To distinguish the input and output arguments in a predicate, we label them with two superscripts, e.g., $P^{k,l}$ corresponds to a predicate of arity $k + l$ where the first k arguments correspond to inputs and the last l to outputs.

Definition 8 (Semantically annotated multi-transducer)**.** *A semantically annotated multi-transducer $T = (\Sigma, S, s_0, f, g, k, l, L_T, L_O, O, U, G)$ is a deterministic transition system with inputs and outputs, where:*

- *Σ is the alphabet (of both inputs and outputs),*
- *S is a non-empty finite set of states,*
- *$s_0 \in S$ is the initial state,*
- *$f : S \times \Sigma^k \longrightarrow S$ is the state transition function,*
- *$g : S \times \Sigma^k \longrightarrow \Sigma^l$ is the output function,*
- *k is the number of input ports and l is the number of output ports,*
- *L_T is a finite set of unary predicates (types of inputs and outputs),*
- *L_O is a finite set of $k + l$-ary predicates (types of operators/transitions),*
- *O is a function from the set of transitions $Tr = \{(s, \mathbf{a}, s', \mathbf{b}) \mid f(s, \mathbf{a}) = s', g(s, \mathbf{a}) = \mathbf{b}\}$ to L_O*
- *$U : Tr \longrightarrow 2^{{L_T}^k}$ and $G : Tr \longrightarrow 2^{{L_T}^l}$ annotate inputs and outputs of $tr \in Tr$.*

We assume that transducers have a distinguished state s_{err} that takes care of incorrect inputs. In the interests of readability, we sometimes omit the s_{err} state in the examples below. We also only use examples of transducers with a single state (not counting s_{err}). Some tools and resources used in workflows are more naturally modelled as a multi-state transducer. For example, the Google Maps Geocoding API will return an error if more than 50 requests are submitted in a second, and modelling such a request counter requires multiple states. However, this is a modelling choice, and any multi-state transducer can be simulated by a finite set of single-state transducers connected by a port binding (introduced below).

We use transducers (representing tools) to generate state transition systems corresponding to a particular instantiation of a workflow. In what follows we essentially treat symbols from Σ as placeholders for concrete data objects (such as specific files etc.) that are manipulated by the tools represented by the transducers. Only

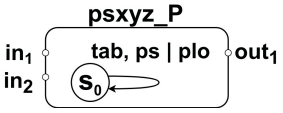


Figure 3.3: Example multi-transducer that performs the *psxyz_P* operation.

(data) objects that satisfy the properties assigned by $U(t)$ result in transitions to a state other than s_{err}, and where the corresponding outputs satisfy the properties assigned by $G(t)$. When objects of the appropriate type are given as inputs (substituted for the symbols) to a transducer transition, new objects are produced with the properties specified for the output.

For example, the semantically annotated transducer in Figure 3.3 corresponds to functional annotations for the operation *psxyz_P* used in the introductory example. The operation can be modelled as $T = (\Sigma, S, s_0, f, g, 2, 1, L_T, L_O, O, U, G)$, where $\Sigma = \{tab, ps, plo, err\}$, $S = \{s_0, s_{err}\}$, $f(s_0, (tab, ps)) = s_0$ and $g(s_0, (tab, ps)) = plo$; all other transitions lead to s_{err} and output err. The types language is $L_T = \{XYZ\_table, PostScript\}$, while the set of operations/transitions is $L_O = \{psxyz\_P^{2,1}(tab, ps, plo)\}$. The annotations for the only meaningful transition are: $O(s_0, (tab, ps), s_0, plo) = psxyz\_P^{2,1}(tab, ps, plo)$, $U(s_0, (tab, ps), s_0, plo) = (\{XYZ\_table\}, \{PostScript\})\}$, $G((s_0, (tab, ps), s_0, plo) = \{PostScript\}$.

A workflow consists of a number of tools connected together. We model this as a **port binding** of a set of transducers. For a multi-transducer $T^x = (\Sigma, S^x, s_0^x, f^x, g^x, k^x, l^x, {L_T}^x, {L_O}^x, O^x, U^x, G^x)$, the input port $1 \leq i < k^x$ is denoted by $in_{x,i}$, and the output port $1 \leq j < l^x$ by $out_{x,j}$. The values at the input port i and output port j of transducer T^x are denoted as $val(in_{x,i})$ and $val(out_{x,j})$, respectively. The values reflect the type of data that is required as input/provided as output. Similarly, $val(in_x)$ and $val(out_x)$ denote the vectors of values at the input and output ports of T^x. The domain of the val function is $L_T \cup \{\epsilon\}$, where ϵ denotes the empty type, i.e., the port requires (input port) or creates (output port) no data. As in [32], we use index $x = 0$ to denote the inputs and outputs of the environment. That is, transducer T^0 specifies the inputs to the workflow and the required outputs: the outputs of the environment are the initial inputs to the set of transducers representing computational tools, $T^1, \ldots, T^m$, and the inputs to the environment are outputs of $T^1, \ldots, T^m$.

Definition 9 (Port binding)**.** *Given a set of semantically-annotated multi-transducers $T^0, \ldots, T^m$, a **port binding** c is a set of pairs of the form $(out_{y,j}, in_{x,i})$ (where $x, y \in \{0, \ldots, m\}$ and i, j are port numbers in $\{1, \ldots, k^x\}$ and $\{1, \ldots, l^y\}$, respectively) that represent connections between the output port j of multi-transducer y and input port i of multi-transducer x. A **workflow port binding** in addition satisfies the following constraints:*

- *each input port is connected to at most one output port (if, for some $i \in \{1, \ldots, k^x\}$, $in_{x,i}$ does not appear in c, its value is assumed to be empty, i.e., $val(in_{x,i}) = \epsilon$);*
- *there are no loops, i.e., there is no path along the edges[1] which is either port*

[1]Each pair $(out_{x,j}, in_{y,i})$ in a binding can be seen as an edge between x and y. A path $x_1, \ldots, x_n$

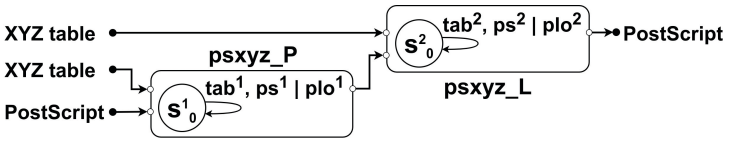


Figure 3.4: A workflow modelled as a port binding between two transducers.

bindings or links between input and output ports of the same transducer in $T^1, \ldots, T^m$.

A port binding together with a set of inputs generates a finite labelled state transition system (a finite path).

Definition 10 (STS generated by a port binding)**.** *Given a set of transducers* $T^0, \ldots, T^m$*, a port binding* c *and an input tuple* $\mathbf{a} = a_1, \ldots, a_n$ *annotated with* $A_1, \ldots, A_n$ *where* $A_j \subseteq L_T$ *for* $1 \leq j \leq n$*, the transducers will make a finite number of transitions* $tr_1 \ldots, tr_k$ *for some* $k \leq m$*, giving rise to a state transition system (path)* π*:*

$$q_0, P_1(\mathbf{a_0}, \mathbf{b_1}), q_1, P_2(\mathbf{a_1}, \mathbf{b_2}), \ldots, P_k(\mathbf{a_{k-1}}, \mathbf{b_k}), q_k$$

where

- $q_0, \ldots, q_k$ *are sets of ground atomic formulas (states of* π*);*
- *transitions between states* q_i *and* q_{i+1} *are labelled by the operator application* $P_{i+1}(\mathbf{a_i}, \mathbf{b_{i+1}})$
- $P_i \in O(tr_i)$*,*
- $\mathbf{a_{i-1}}$ *are inputs and* $\mathbf{b_i}$ *(fresh constants) are outputs of* tr_i*,*
- $\mathbf{a_0} = \mathbf{a}$ *are the values on the output port of the environment,*
- $q_0 = \bigcup_{j \in \{1,\ldots,n\}} \{P(a_j) \mid P \in A_j\}$*,*
- $q_i = q_{i-1} \cup \{P(b) \mid \exists j (b = \mathbf{b_i}_j, P \in G(tr_i)_j)\}$.

Figure 3.4 illustrates a port binding between two transducers that can plot points (*psxyz_P*) and lines (*psxyz_L*). If the environment is modelled as transducer 0, (*psxyz_P*) as transducer 1 and (*psxyz_L*) as transducer 2, the binding is as follows: $\{(out_{0,1}, in_{2,1}), (out_{0,2}, in_{1,1}), (out_{0,3}, in_{1,2}),$
$(out_{1,1}, in_{2,2}), (out_{2,1}, in_{0,1})\}$. Given specific input files a, b and c, this binding generates the following state transition system:

along the edges in a port binding exists if some output port of x_1 is connected to some input port of x_2, an output port of x_2 is connected to an input port of x_3,..., and an output port of x_{n-1} is connected to an input port of x_n

$$
\begin{aligned}
q_0 &= \{XYZ\_Table(a), XYZ\_Table(b), PostScript(c)\} \\
o_1 &= psxyz\_P^{2,1}(b,c,d) \\
q_1 &= \{XYZ\_Table(a), XYZ\_Table(b), PostScript(c), \\
&\qquad PostScript(d)\} \\
o_2 &= psxyz\_L^{2,1}(a,d,e) \\
q_2 &= \{XYZ\_Table(a), XYZ\_Table(b), PostScript(c), \\
&\qquad PostScript(d), PostScript(e)\}
\end{aligned}
$$

3.3 SLTL^x

In this section, we extend Semantic Linear Temporal Logic (SLTL) [130] to be able to talk about (data) objects. We call the resulting logic SLTL^x.

Similar to SLTL, SLTL^x presupposes the existence of semantic type hierarchies. The low level operation names and signatures, such as $psxyz\_P^{2,1}(x,y,z)$, are unlikely to be known to the users who specify a workflow. Users are more likely to use a high-level specification of an operation, such as $Plot\_Points^{1,1}(u,v)$ (where u is the input file with coordinates and v is the output map). This necessitates including more operation names in the specification language than those corresponding to the transducers, and a representation of a relationship between concrete and abstract operations. In addition, a user may also specify properties of files which are not in the standard type hierarchy, such as $Birds(a)$ to say that file a contains data on the movements of birds.

Unlike SLTL, SLTL^x introduces a distinguished binary predicate R to track 'ancestor relations' between objects. An object a is an ancestor of object b, $R(a,b)$, if either $a = b$ or b is an output of an operation that had as one of the inputs an object a' such that $R(a,a')$ (note that R is a transitive relation). While the user may not know the order of operations and the required types of their inputs, they may want to specify that an operation should be performed either directly on the input file or on a file derived from it. For example, a user may require that $Plot\_Points^{1,1}(u,v)$ should be applied either to a file containing coordinates of cities ($Cities(u)$) or to a file that has been obtained from u by performing some processing, $Cities(w) \wedge R(u,w)$.

The syntax of SLTL^x is defined relative to the following alphabet:

- a countable set of variables $Var = \{x, y, z, \ldots\}$,
- a countable set of constants $Con = \{a, b, c, \ldots\}$,
- a finite set L^t of unary predicate symbols that includes L_T,
- a finite set L^o of predicates symbols that includes L_O,
- a distinguished binary predicate R,
- identity relation between terms, $=$,
- propositional connectives $true$, $\neg$, $\wedge$,
- temporal operators $\mathbf{G}$ (always in the future) and $\mathbf{U}$ (until), and dynamic operators $\langle P^{k,l}(t_1, \ldots, t_{k+l})\rangle$, $P^{k,l} \in L^o$.

Terms are variables or constants, where each constant depicts a (data) object in a workflow execution. Atomic formulas are of the form $P(t_1)$, where $P \in L^t$, $R(t_1, t_2)$ or $t_1 = t_2$, where P and R are a unary and a binary predicate, respectively, and t_1 and t_2 are terms.

The set of ground atomic formulas built using types L^t will be denoted by At^{L^t}. The states will be subsets of At^{L^t}. The set of ground atoms constructed using 'concrete' operators L_O will be denoted by At^{L_O}. Transitions between states correspond to elements of At^{L_O} (we assume that there are no parallel operations by two or more transducers).

We define an 'implements' relation $\triangleright$ between atomic formulas over At^{L_O} and formulas built using L^o to say that a description of a concrete operation is an implementation of an abstract one. This relation is derived from semantic hierarchies for a particular domain. For example, $psxyz\_P^{2,1}(a, b, c)$ implements $Draw\_Points^{1,1}(a, c)$, symbolically, $psxyz\_P^{2,1}(a, b, c) \triangleright Draw\_Points^{1,1}(a, c)$.

The syntax of SLTL^x is given by the following BNF:

$$\Phi ::= true \mid P(t) \mid R(t_1, t_2) \mid \neg \Phi \mid \Phi \wedge \Phi \mid$$
$$\langle P(t_1, \ldots, t_n) \rangle \Phi \mid \mathbf{G}\, \Phi \mid \Phi\, \mathbf{U}\, \Phi \mid \exists x \Phi \mid t_1 = t_2$$

$P(t)$ depicts property of a term t, where $P \in L^t$, $\langle P(t_1, \ldots, t_n) \rangle \Phi$ means 'after applying operation $P(t_1, \ldots, t_n)$, Φ holds', $\mathbf{G}$ and $\mathbf{U}$ are 'globally' and 'until'.

Given a set of ground atoms A, we define the domain of A, $dom(A)$, to be the set of all constants occurring in A. An **assignment** θ on A is a function from the set of variables, Var into $dom(A)$. For a term t, $[t]_\theta = t$ if t is a constant, and $\theta(t)$ if t is a variable. Sentences of SLTL^x are formulas with no free variables. Workflows are specified by sentences of SLTL^x.

An SLTL^x model $\pi = (q_0, o_1, q_1, o_2, q_2, ..., q_{k-1}, o_k, q_k)$ is a finite alternating sequence of states (subsets of At^{L^t}) and ground transition relations (elements of At^{L_O}). We denote by π_i, $0 \leq i < k$, the suffix $q_i, o_{i+1}, \ldots, q_k$ of π; for $i = k$, $\pi_k = (q_k)$. We denote by θ_i, $0 \leq i < k$, an assignment over $q_i \cup \{o_{i+1}\}$, and θ_k an assignment over q_k. Note that given a state q_i in π, it is possible to compute the reflexive and transitive relation R on $dom(q_i)$ from $o_1, \ldots, o_i$.

Definition 11 (Truth conditions in SLTL^x models)**.** *Let* $\pi = (q_0, o_1, q_1, o_2, q_2, ..., q_{k-1}, o_k, q_k)$ *be an SLTL^x model. The relation "*π *satisfies formula* Φ *under as-*

signment θ" ($\pi, \theta \vDash \Phi$) as $\pi_0, \theta_0 \vDash \Phi$ by induction below:

$$
\begin{array}{lll}
\pi_i, \theta_i \vDash true & & \\
\pi_i, \theta_i \vDash P(t) & \textit{iff} & P([t]_{\theta_i}) \in q_i \\
\pi_i, \theta_i \vDash t_1 = t_2 & \textit{iff} & [t_1]_{\theta_i} = [t_2]_{\theta_i} \\
\pi_i, \theta_i \vDash R(t_1, t_2) & \textit{iff} & R([t_1]_{\theta_i}, [t_2]_{\theta_i}) \\
\pi_i, \theta_i \vDash \neg\Phi & \textit{iff} & \pi_i, \theta_i \nvDash \Phi \\
\pi_i, \theta_i \vDash \Phi_1 \wedge \Phi_2 & \textit{iff} & \pi_i, \theta_i \vDash \Phi_1 \wedge \pi_i, \theta_i \vDash \Phi_2 \\
\pi_i, \theta_i \vDash \exists x \Phi & \textit{iff} & \exists d \in dom(q_i \cup \{o_{i+1}\})\ (\pi_i, \theta_i[x \mapsto d] \vDash \Phi) \\
\pi_i, \theta_i \vDash \langle P(t_1, \ldots, t_n)\rangle\Phi & \textit{iff} & o_1 \rhd P([t_1]_{\theta_i}, \ldots, [t_n]_{\theta_i}) \\
 & & \textit{and } \pi_{i+1}, \theta_{i+1} \vDash \Phi \textit{ and } k > 0 \\
\pi_i, \theta_i \vDash \boldsymbol{G}\Phi & \textit{iff} & \forall j \in \{i, ..., k\} : \pi_j, \theta_j \vDash \Phi \\
\pi_i, \theta_i \vDash \Phi_1 \boldsymbol{U} \Phi_2 & \textit{iff} & \exists j \in \{i, ..., k\} : \\
 & & \forall m \in \{i, ..., j-1\} : \\
 & & \pi_m, \theta_m \vDash \Phi_1 \textit{ and } \pi_j, \theta_j \vDash \Phi_2
\end{array}
$$

We use the standard definitions for $\vee$ and $\rightarrow$. In addition to the $\boldsymbol{G}$ and $\boldsymbol{U}$ operators as defined above, we will use two additional operators to simplify notation: $\boldsymbol{X}\Phi$, interpreted as $\langle true\rangle\Phi$, denotes the *next-time* operator, and $\boldsymbol{F}\Phi$, interpreted as $true\,\boldsymbol{U}\,\Phi$, denotes *eventually* operator. Note that although we can refer to transitions, the logic is much closer to LTL on finite traces (LTL$_f$) than to Linear Dynamic Logic on finite traces LDL$_f$ in [31].

3.4 Transducer Synthesis with Temporal Goals

In this section, we define three workflow synthesis problems and analyse their complexity. The three synthesis problems differ in expressive power, applicability as well as complexity, and thus, are presented individually.

Definition 12 (Bounded workflow synthesis)**.** *The **bounded workflow synthesis problem** is: given a set of semantically annotated multi-transducers, $T^1, \ldots, T^m$, an SLTL^x formula Φ (the goal formula), and an initial input tuple* $\mathbf{a}$*, is there a port binding for some subset of $T^1, \ldots, T^m$ such that the resulting SLTL^x model satisfies Φ.*

Theorem 1. *The bounded workflow synthesis problem is NP-complete.*

Proof. For membership in NP, observe that a port binding, for a fixed set of transducers and input objects, is polynomial in the size of the problem input. Hence it is possible to guess a port binding, generate the corresponding state sequence (which is of finite length polynomial in the input since there are no cycles in the binding), and check whether it satisfies the formula Φ in polynomial time. This means that the problem can be solved by a non-deterministic Turing machine in polynomial time.

For NP-hardness, we use a reduction from the satisfiability of CNF formulas. Let ϕ be a CNF formula over variables $p_1, \ldots, p_n$. The reduction is as follows. The set of n transducers contains, for each p_i, a p_i-transducer that on input x outputs y that

has property P_i. Let $tr(\phi)$ be a translation of ϕ into first-order logic that replaces p_i with $\exists x P_i(x)$ and $\neg p_i$ with $\forall x \neg P_i(x)$. (Since ϕ is in CNF, all negations occur only on propositional variables.) Then ϕ is satisfiable iff there is a positive answer to the bounded workflow synthesis problem for this set of transducers, an input a with an empty set of annotations, and a goal formula $\boldsymbol{F}tr(\phi)$. □

In bounded workflow synthesis, the workflow is restricted to the specified set of tools represented by $T^1, \ldots, T^m$ (some of which could be copies of the same tool). Recall that workflows are acyclic, so each T^i can be used at most once in the workflow. However, it is often not possible to specify how many copies of a given transducer may be needed. For example, a user may not know in advance how many times e.g., a postscript generator will need to be used and hence how many copies of the postscript generator transducer to specify.

Definition 13 (Unbounded workflow synthesis)**.** *The **unbounded workflow synthesis problem** is: given a set of semantically annotated multi-transducers, $T^1, \ldots, T^m$, an SLTL^x formula Φ (the goal formula), and an initial input tuple $\mathbf{a}$, are there non-negative integers $n_1, \ldots, n_m$ is there a port binding for n_1 copies of T^1, ..., n_m copies of T^m, such that the resulting SLTL^x model satisfies Φ.*

We show in [72] that the unbounded workflow synthesis problem is undecidable for $m \geq 1$. However, for a subset of goal formulas, a slight modification of the unbounded synthesis problem is decidable.

Definition 14 (Feasible goal formulas)**.** *A* feasible goal formula *is an SLTL^x formula $\varphi = \boldsymbol{F}(\phi_1 \wedge \ldots \wedge \phi_v)$ where each ϕ_i is either of the form*

- $\exists x_i \psi(x_i)$ *where $\psi(x_i)$ is a boolean combination of atoms $P(x_i)$ with $P \in L^t$ or*
- $\exists y_1 \ldots y_n \langle P(y_1, \ldots, y_n) \rangle true$

Instead of a fixed binding of copies of transducers, we construct a **dynamic binding**. Intuitively, now the orchestrator is going to construct a new port binding after each transition by the transducers, collect the outputs, and construct a binding again. A useful intuition may be to think of the orchestrator as a planner and of the transducers as operator schemas. The difference from classical planning is as follows: we do not know all the objects in advance, since new objects can be created; properties of objects are not changed once they are established; and the properties of objects in the goal formula are all unary. We can even specify which operators should be used in the workflow, although without specifying how their arguments relate to other terms in the formula.

The difference from the unbounded orchestration problem is that the 'width' of binding constructed at each step is restricted to using only one copy of each transducer at a single time (e.g., we do not take to copies of T^1 and bind the output ports of the environment to the input ports of both copies).

Definition 15 (Dynamic workflow synthesis)**.** *The **dynamic** workflow synthesis problem is as follows: given a finite set of semantically annotated multi-transducers $T^1, \ldots, T^m$, a feasible goal formula $\boldsymbol{F}(\phi_1 \wedge \ldots \wedge \phi_v)$, and an initial input tuple $\mathbf{a}$, is there a sequence of port bindings such that the initial port binding allocates elements from $\mathbf{a}$ to some input ports of a subset of $T^1, \ldots, T^m$, and each subsequent binding allocates outputs from the previous step to (possibly different) input ports of a (possibly*

different) subset of $T^1, \ldots, T^m$, and the transition system generated by the transducers under this sequence of bindings satisfies the goal formula.

Lemma 1. *The dynamic SLTL^x workflow synthesis problem is PSPACE-hard.*

Proof. The proof is by reduction from STRIPS planning. An instance of a propositional STRIPS planning problem as defined in [Bylander 1994] is a tuple $(\mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G})$ where

- $\mathcal{P}$ is a finite set of ground atomic formulas, called the conditions;
- $\mathcal{O}$ is a finite set of operators, where each operator o has the form $Pre \Rightarrow Post$:
  - Pre consists of two disjoint subsets of $\mathcal{P}$, called positive preconditions o^+ and negative preconditions o^- of the operator, such that the conjunction of o^+ and negated o^- conditions is satisfiable;
  - $Post$ consists of two disjoint subsets of $\mathcal{P}$, called positive postconditions o_+ and negative postconditions o_- of the operator, such that the conjunction of o_+ and negated o_- conditions is satisfiable;
- $\mathcal{I} \subseteq \mathcal{P}$ is the initial state; and
- $\mathcal{G}$, the goal condition, consists of two disjoint subsets of $\mathcal{P}$, called positive goals $\mathcal{G}_+$ and negative goals $\mathcal{G}_-$, such that the conjunction of $\mathcal{G}_+$ and negations of conditions in $\mathcal{G}_-$ is satisfiable.

The effect of a finite sequence of operators $(o_1, \ldots, o_n)$ on a state s is formalised as follows:

- $Res(s, ()) = s$
- If $o^+ \subseteq s$ and $o_- \cap s = \varnothing$, $Res(s, (o)) = (s \cup o_+) \setminus o_-$; otherwise $Res(s, (o)) = s$.
- $Res(s, (o_1, \ldots, o_n)) = Res(Res(s, o_1), (o_2, \ldots, o_n))$

$(o_1, \ldots, o_n)$ is a solution to an instance of propositional STRIPS planning problem if $Res(\mathcal{I}, (o_1, \ldots, o_n))$ is a goal state, that is, $\mathcal{G}_+ \subseteq Res(\mathcal{I}, (o_1, \ldots, o_n))$ and $\mathcal{G}_- \cap Res(\mathcal{I}, (o_1, \ldots, o_n)) = \varnothing$. An instance of propositional STRIPS planning problem is satisfiable if has a solution.

The idea of the reduction is as follows. Given $(\mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G})$, for each operator in $\mathcal{O}$, there is a transducer with one input port and one output port. The inputs and outputs of the transducers correspond to the states in the planning process. Conditions in $\mathcal{P}$ become unary predicates which are annotations of inputs and outputs. The initial input is annotated with properties of $\mathcal{I}$, and the goal formula asserts that in the future there is an output annotated with properties of $\mathcal{G}$.

Let $\mathcal{O} = \{o_1, \ldots, o_m\}$ and $\mathcal{P} = \{p_1, \ldots, p_r\}$. Then $L^o = \{O_i(x, y) : o_i \in \mathcal{O}\}$ and $L^t = \{P_1, \ldots, P_r\}$. We define a translation function tr_x from propositional to first order formulas with variable x, as follows:

- $tr_x(p_i) = P_i(x)$
- $tr_x(\neg p_i) = \neg P_i(x)$
- $tr_x(\phi_1 \wedge \ldots \wedge \phi_k) = tr_x(\phi_1) \wedge \ldots \wedge tr_x(\phi_k)$

For simplicity, we identify sets of formulas with conjunctions of those formulas when appropriate. We also identify a set of unary predicates annotating a variable z with a set of atomic formulas in z using those predicates. We introduce the following notation: for an object z, the set of L^t formulas it is annotated with is denoted by $ann(z)$.

For a set of atoms Z of the form $P_i(z)$, where $P_i \in L^t$, we denote by $tr_z^{-1}(Z)$ the

set $\{p_i \in \mathcal{P} : tr_z(p_i) \in Z\}$.

Given an instance of propositional planning problem $(\mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G})$, we generate an instance of a dynamic SLTL^x workflow synthesis problem $(T^1, \ldots, T^m, a, \phi)$ where:

- each T^i corresponds to $o_i \in O$. Each T_i has one input port and one output port, and has one transition $t^i = (s_0^i, x^i, s_0^i, y^i)$ that takes one input x^i and outputs y^i.
  - ∴ $U(t^i) = tr_{x^i}(o_i^+)$
  - ∴ if $tr_{x^i}(o_i^-) \cap ann(x^i) = \varnothing$, $G(t^i) = tr_{y^i}(Res(tr_{x^i}^{-1}(ann(x^i))), o_i))$ (intuitively, if the input x^i is annotated with $tr_{x^i}(s)$, and its annotations satisfy the preconditions of o_i, then $G(t^i) = tr_{y^i}(Res(s, o_i))$. Else $G(t^i) = ann(x^i)[x^i/y^i]$.
- $ann(a) = tr_a(\mathcal{I})$
- the goal formula is $\exists z\ \mathbf{F}\ tr_z(\mathcal{G}_+ \wedge \bigwedge_{g_- \in \mathcal{G}_-} \neg g_-)$.

Clearly, the reduction is polynomial in the size of $(\mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G})$.

To show that if $(\mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G})$ is satisfiable then the corresponding dynamic SLTL^x workflow synthesis problem has a solution, assume that there is a sequence of operators $(o_1, \ldots, o_n)$ such that $Res(\mathcal{I}, (o_1, \ldots, o_n))$ satisfies $\mathcal{G}$. Then there is a sequence of bindings $(out_{0,1}, in_{1,1})$; $(out_{1,1}, in_{2,1})$; …; $(out_{n,1}, in_{0,1})$ that on input a satisfying $tr_a(\mathcal{I})$ eventually produces output b satisfying $tr_b(\mathcal{G})$. The corresponding state transition system satisfies $\exists z\ \mathbf{F}\ tr_z(\mathcal{G}_+ \wedge \bigwedge_{g_- \in \mathcal{G}_-} \neg g_-)$.

Conversely, if there is a sequence of bindings $(out_{0,1}, in_{1,1})$; $(out_{1,1}, in_{2,1})$; …; $(out_{n,1}, in_{0,1})$ that on input a satisfying $tr_a(\mathcal{I})$ produces a state transition system satisfying $\exists z\ \mathbf{F}\ tr_z(\mathcal{G}_+ \wedge \bigwedge_{g_- \in \mathcal{G}_-} \neg g_-)$, this means that an output b generated by one of the bindings satisfies $tr_b(\mathcal{G})$. Hence the corresponding sequence of planning operators transforms the state $\mathcal{I}$ into a goal state satisfying $\mathcal{G}$. □

Theorem 2. *The dynamic workflow synthesis problem is PSPACE-complete.*

Proof. We first introduce some notation and terminology. Let us denote by $K = \Sigma_{j=1,\ldots,m} k^m$ the maximal number of input ports that can be used simultaneously in parallel.

Observe that the set L^t is finite; a complete description of an object in terms of the types in L^t is a conjunction of atoms and negated atoms for each type $P \in L^t$. There are $2^{|L^t|}$ such complete descriptions, which we will refer to as supertypes.

Let us consider first the case where the goal formula is of the form $\boldsymbol{F}\exists x \psi(x)$. If an object satisfying ψ can be constructed at all, there is a sequence of states leading to a state which contains an object satisfying ψ. This sequence does not have repetitions. Each state can be uniquely described as an allocation of one of $2^{|L^t|}$ supertypes to each of possible K input ports (the outputs are produced deterministically), so there are $2^{|L^t| \times K}$ different states. Clearly, the sequence leading from the initial state to a $\psi(x)$ state can be exponentially long. However, similarly to classical planning, a state can be represented in polynomial space by listing at most $|L^t|$ positive properties for each of K input ports. A path-exists(q_1, q_2, N) algorithm that checks the existence of a path of length N between states q_1 and q_2 (where q_2 satisfies the goal test, that is, outputs a $\psi(x)$ object) requires polynomial space; for $N = 1$ it checks whether $q_1 = q_2$ or there is a single step transition between them; for $N > 1$ it re-

cursively calls plan-exists($q_1, q_3, \lceil N/2 \rceil$) and plan-exists($q_3, q_2, \lfloor N/2 \rfloor$). Note that N represented in binary takes $O(logN)$ space, so is polynomial in the input size [22].

To check whether the required operators have been used, we can modify the path-exists(q_1, q_2, N) algorithm to return the set of operator names encountered on the path from q_1 to q_2. Observe that this set of names (unlike the complete list of all ground operator formulas on the path) is polynomial in the input size.

The problem of generating several objects with specified types is no harder than for a single object, because properties of objects persist.

PSPACE-hardness is shown by Lemma 1. □

3.5 Evaluation and Discussion

The section presents benefits of the new SLTL^x-based formalism, when compared to the existing SLTL-based one. The evaluation of the usability of the framework is further evaluated in Chapter 6. Our implementation of the approach (APE v2) is used to explore new solutions to existing problems in life- and geo-science domains. The case studies include domain expert opinions and evaluations of the composed solutions.

We show, using recent case studies, how the SLTL^x-based formalism overcomes the limitations of the SLTL-based approach described in this chapter. We use the most recent implementations of each of the formalisms, APE v2 and APE v1, respectively, to run the synthesis.

The evaluation focuses on examples from recent case studies in Geosciences and the corresponding domains. Each of the use cases is further evaluated according to the following criteria.

- *Optimal solutions* - Solutions of the synthesis are *optimal* if no redundant or incorrect steps are included.
- *Unique model* - Each obtained temporal logic model identifies a single workflow implementation. A model is not unique if it can be interpreted as two or more workflow implementations.
- *Fully automated* - A synthesis of a problem is *fully automated* if it does not require any manual steps apart from providing the temporal specification of the problem.

We focus on benefits of the SLTL^x-based approach compared to the existing SLTL-based one. To accomplish that, we evaluate the results by comparing them to concrete target workflows. We are evaluating specific features of the framework, rather than the applicability of the synthesis approach. The latter is covered in Chapter 6.

3.5.1 Geovisualisation

The example given in the introductory section illustrates the limitations of the SLTL-based formalism when it comes to data object distinction. The example is in fact a fragment of a larger geovisualisation case study [73], that aims to synthesise workflows that visualise bird movements in the Netherlands with respect to the regional topography. The synthesis is performed over a set of GMT[2] (The Generic Mapping

[2]https://www.generic-mapping-tools.org/

Tools) operations, annotated with respect to their inputs and outputs, and an ontology that classifies the utilised data types and operations. The operations are modelled as single state multi-transducers, and thus, a simple input/output annotation is sufficient. For example transducers from Figure 3.3 is modelled as an operation *psxyz_P* with two inputs (*XYZ_Table* and *PostScript*) and one output (*PostScript*). The original encoding of the problem comprises multiple files of the same format/type as input. This reflects similar scenarios in other domains and makes an interesting evaluation.

To encode such a problem using an SLTL-based framework, the specification must be manually divided into sub-problems that do not contain multiple objects of the same data type. Subsequently, the resulting sub-workflows must be manually combined into the target workflow. In the geovisulisation case study, the presented approach was used to split the target workflow into four target sub-workflows that were solved separately using the SLTL-based approach. The shortest solution to the problem is a workflow of length 17.

Notice that whether the problem can be divided into valid sub-problems under the SLTL formalism, depends on the problem specification. For example, a domain model that contains tools with multiple inputs of the same type might not support such an approach. To illustrate, if we want to combine data tracking various flocks of the tracked bird species, we have to use a tool that would combine the tracking files into one. The corresponding workflow must include such operation with two (or more) input files of the same type/format, and thus, the corresponding workflow fragment is not expressible in SLTL even with manual workarounds.

The SLTL^x-based formalism, however, captures such specifications with a single formula that can be fed to the SLTL^x synthesis engine, APE v2 in our case. Considering that the explanatory (see Figure 3.1) and the complete target workflows have similar structures, we illustrate the encoding on the explanatory case before we present the complete encoding. The specification can be formalised in SLTL^x as follows:

$$
\begin{aligned}
\Phi_1 = {} & CSV(a) \wedge CSV(b) \wedge Cities(a) \wedge Birds(b) \wedge \boldsymbol{F}\, \exists x_1 \langle Plot\_water^{0,1}(x_1)\rangle \\
& (\boldsymbol{F}\, \exists x_2 \langle Plot\_coast^{0,1}(x_2)\rangle (\boldsymbol{F}\, \exists x_3, \exists y (R(a,y) \wedge \langle Plot\_points^{1,1}(y,x_3)\rangle \\
& (\boldsymbol{F}\, \exists x_4, \exists z (R(b,z) \wedge \langle Plot\_lines^{1,1}(z,x_4)\rangle true))))) \wedge \boldsymbol{F}\, \exists x_5 \langle Tool^{0,1}(x_5)\rangle \\
& (PostScript(x_5) \wedge R(x_1,x_5) \wedge R(x_2,x_5) \wedge R(x_3,x_5) \wedge R(x_4,x_5) \wedge \neg \boldsymbol{X} true))
\end{aligned}
$$

It ensures that bird coordinates (labelled $Birds(b)$) are connected by lines, while city coordinates (labelled $Cities(a)$) are depicted as points. As a result, the workflow interpretations in Figure 3.1(a)-(d) can be distinguished, and only the workflows satisfying the specification will be synthesised. For this workflow fragment, APE v2 generates exactly one workflow of length 7, which corresponds to the desired workflow in Figure 3.1(c).

Similarly, we use SLTL^x to encode the full target workflow used in the [73] case study as follows:

Use Case		Optimal solutions	Unique model	Fully automated
Geovisualisation	SLTL	✓*		
	SLTL^x	✓	✓	✓
GIS Question Answering	SLTL			✓
	SLTL^x	✓	✓	✓

Table 3.1: Comparison of the SLTL- and SLTL^x-based synthesis approaches in the Geovisualisation and GIS Question Answering case studies (*assuming that the specified problem is divisible in sub-problems expressible under SLTL).

$$
\begin{aligned}
&CSV(a) \wedge CSV(b) \wedge Cities(a) \wedge Birds(b) \wedge \boldsymbol{F}\, \exists x_1 \langle Draw\_boundary\_frame^{0,1}(x_1)\rangle true \\
&\wedge \boldsymbol{F}\, \exists x_2 \langle Add\_table^{0,1}(x_2)\rangle (\boldsymbol{F}\, \exists x_3, \exists x_4\, Colour\_palette(x_3) \langle 2D\_surfaces^{2,1}(x_2, x_3, x_4)\rangle \\
&(\boldsymbol{F}\, \exists x_5 \langle Gradient\_generation^{0,1}(x_5)\rangle (\boldsymbol{F}\, \exists x_6, \exists x_7\, Colour\_palette(x_6) \\
&\langle 2D\_surfaces^{1,1}(x_6, x_7)\rangle (R(x_3, x_6) \wedge \neg(x_3 = x_6) \wedge (\boldsymbol{F}\, \exists x_8, \exists x_9 \\
&\langle Draw\_color\_range^{1,1}(x_8, x_9)\rangle (R(x_6, x_8) \wedge (\boldsymbol{F}\, \exists x_{10}, \exists y\, (R(a, y) \wedge \\
&\langle Plot\_points^{1,1}(y, x_{10})\rangle (\boldsymbol{F}\, \exists x_{11}, \exists z (R(b, z) \wedge \langle Plot\_lines^{1,1}(z, x_{11})\rangle \\
&(\boldsymbol{F}\, \exists x_{12} \langle Draw\_political\_borders^{0,1}(x_{12})\rangle true)))))))))) \\
&\wedge \boldsymbol{F}\, \exists x_{13} \langle Tool^{0,1}(x_{13})\rangle (PostScript(x_{13}) \wedge R(x_1, x_{13}) \wedge R(x_2, x_{13}) \\
&\wedge R(x_3, x_{13}) \wedge R(x_4, x_{13}) \wedge R(x_5, x_{13}) \wedge R(x_6, x_{13}) \wedge R(x_7, x_{13}) \wedge R(x_8, x_{13}) \\
&\wedge R(x_9, x_{13}) \wedge R(x_{10}, x_{13}) \wedge R(x_{11}, x_{13}) \wedge R(x_{12}, x_{13}) \wedge \neg \boldsymbol{X} true))
\end{aligned}
$$

APE v2 is able to synthesise the full 17-step workflow for this case study in a fully automated fashion. The evaluation according to the three criteria presented earlier, is presented in Table 3.1.

We notice that the SLTL-based approach requires manual steps to split the problem into solvable fragments, and to combine the solved fragments into an executable workflow. The SLTL^x-based framework supports specifying the existence of multiple data objects (files) characterised by the same type of data and format. This allows us to encode more accurately the user intent and automate the composition of a much wider range of workflows. Furthermore, we have seen an example of a specification that is not solvable by the SLTL-based approach even with the manual fragmentation of the problem.

Finally, unlike the SLTL^x model that can be directly translated into a workflow implementation, the SLTL model does not preserve data dependencies. Therefore, SLTL models require post-processing to be translated into a workflow implementation. This process however is not always straightforward and solutions that contain data repetitions, such as the presented one, cannot be uniquely implemented. In such cases further steps are needed to determine the implementation that fits the users' intent.

All the data used to run the case study and generate the results is available at `https://github.com/sanctuuary/APE_UseCases/tree/master/GeoGMT`.

3.5.2 Geo-Analytical Question Answering

In the introduction we mentioned the Geo-Analytical Question Answering case study [84], where APE v1 was used to automatically synthesise workflows that answer given livability questions (such as "What is the accessibility of parks for each administrative region in Amsterdam?"). The study shows quite promising results, but it also points out a limitation of the underlying formalism. It shows that around 70% of the 72 synthesised workflows[3] contain redundant operations. The reason for that in the majority of the cases is a repetition of transformations over same the data objects. To avoid such occurrences the user would specify a constraint of the form "Do not transform the same data multiple times". The SLTL formalism, however cannot capture such constraints, as it does not express dependencies between operations and data objects. The closest it can get to specifying the constraint is to express constraint of the form "Do not perform more than one transformation" (in SLTL written as $\mathbf{G}\big(\langle Transform\rangle \mathit{true} \Rightarrow \mathbf{XG}\neg\langle Transform\rangle \mathit{true}\big)$). Such constraint is too restrictive for the given scenario as it prevents transformations of any other data object. Therefore such redundancies are unavoidable under the SLTL-based approach, i.e., when using the APE v1 system.

The SLTL^x-based approach behind APE v2 allows us to express data object specific constraints, including their interaction with individual operations, i.e., data-tool dependencies. This allows us to avoid the aforementioned redundancies by preventing more than one transformation over each data object in SLTL^x, as follows.

$$\begin{aligned}\Phi_2 =&\neg(\boldsymbol{F}\exists x_1(\langle Transform^{0,1}(x_1)\rangle \boldsymbol{F}\langle Transform^{1,0}(x_1)\rangle true))\wedge\\ &\neg(\boldsymbol{F}\exists x_1(\langle Transform^{1,0}(x_1)\rangle \boldsymbol{F}\langle Transform^{1,0}(x_1)\rangle true))\end{aligned} \tag{3.1}$$

where the operation *Transform* is a superclass (in the domain taxonomy) of all the operations that perform transformations.

The constraint ensures 1) that the results (outputs) of transformations are not transformed again, and 2) that individual data objects are not transformed more than once. Adding such a constraint to the specification of the geovisualisation case study is sufficient to exclude the detected workflows containing redundant operations from the synthesised workflows.

The evaluation according to the three criteria presented earlier, is presented in Table 3.1. Due to the expressive power of the SLTL formalism, the framework cannot ensure optimal solutions at each desired length of the workflow, as longer solutions tend to introduce many candidate workflows that contain redundant information. The SLTL^x formalism, on the other hand, excludes occurrences of such operations and explores new approaches to processing the data at each length. Both approaches solve the specifications from the case study in a fully automated fashion. However, as some of the candidate solutions create multiple data objects at the same time, the SLTL-based approach cannot guarantee a unique interpretation of the models.

All the data used to run the case study and generate the results is available at `https://github.com/sanctuuary/APE_UseCases/tree/dev/QuAnGIS`.

[3] For each of the five livability question on average 14 different candidate workflows were synthesised.

	SLTL				
	Solutions found	Encoding time (sec)	Solving time (sec)	Max length explored	Average length
Q1	20	1.8	0.8	4	3.5
Q2	8	2.23	0.1	8	4.5
Q3	20	3.85	0.9	6	4.45
Q4	4	4.1	0.4	8	5
Q5	20	1.9	0.8	5	4.05

	SLTL^x				
	Solutions found	Encoding time (sec)	Solving time (sec)	Max length explored	Average length
Q1	20	4.1	0.8	5	4.35
Q2	1	7.1	0.2	8	1
Q3	20	6.8	1.8	6	4.45
Q4	1	7.2	0.3	8	2
Q5	20	4.8	0.9	6	4.55

Table 3.2: Comparison of the SLTL-based (APE v1) and SLTL^x-based approach (APE v2) runtime in the GIS Question Answering case study.

3.5.3 Performance of the new implementation

To evaluate the runtime of each of the formalisms, we focus on their latest implementations, namely, APE v1 and APE v2. APE v2 relies on the SAT-based encoding introduced in APE v1 and expands on it, and thus, the encoding that relies on the constraints supported by both formalism yields equally good synthesis execution times. On the other hand, case studies that utilise SLTL^x-specific features, such as the one presented on Geovisualisation, cannot be fully automated using the APE v1 formalism, and thus the runtime comparison is not possible. That is why we compare the runtime of APE v2-generated optimal solutions over the GIS QA case study, with the APE v1-generated suboptimal solutions. To accomplish that we use the set of constraints defined in the original case study, and in the case of APE v2 we add the additional SLTL^x constraint presented in Formula 3.1. The case study comprises five research questions and, for each, solutions up to length 8 are synthesised (with an upper limit of 20 workflows per question). The question that we evaluate are:

Q1: “What is the number of sports facilities in each PC4 area?”,
Q2: “What is the proportion of elderly people living in each PC4 area in Amsterdam?”,
Q3: “What is the accessibility of parks for each PC4 area in Amsterdam??”,
Q4: “What is the amount of noise pollution in each PC4 area in Amsterdam?” and
Q5: “What is the average temperature within each PC4 area in Amsterdam?”.

The comparison of the runtime of APE v1 and APE v2 formalisms is presented in Table 3.2. All experiments were performed on a PC with a 2.50GHz i7-6500U CPU with 16GB RAM running on Ubuntu 20.04. The recorded times were recorded as

average times out of 10 individual runs.

We notice that the SLTL^x-based approach excludes approximately 70% of the solutions which contain redundant steps in the original study (performed using the SLTL-based approach). This is usually reflected in either fewer solutions (e.g., **Q2** and **Q4** result in only one solution each up to length 8) or longer average length of the solutions (e.g., although **Q1** and **Q5** result in the same number of solutions, the SLTL^x-based approach excludes the shorter solutions and explores longer workflows). The only case where this is not directly visible from the table is **Q3**, where the new solutions of length 6 substitute those of the same length containing redundancies. When it comes to the synthesis time, both approaches yield similar runtime, which shows us that the synthesis over the new encoding in practice performs as well as its predecessor. The downside is that an SLTL^x formula, which allows us to exclude the undesirable solutions (see Formula 3.1), slightly increases the propositional encoding time. That is however expected, as we use arbitrary SLTL^x formulas, while the existing approach (APE v1) restricts constraints to predefined SLTL templates. Such arbitrary SLTL^x formulas suffer from the exponential blowup in size when encoded in propositional form, a problem inherent to even simpler temporal logics (including SLTL). We have however managed to optimise the encoding, to provide an implementation that in practice substantially reduces the mentioned problems.

3.6 Related Work

This section presents an overview of the approaches that tackle the synthesis of computational components, often described using input/output dependencies.

Pnueli and Rosner [114] prove that the synthesis of distributed finite-state controllers for a given specification is undecidable. Furthermore, Lustig and Vardi [97] show that the synthesis of component libraries for data-flow composition, where components are chained together w.r.t. their outputs and inputs, is also undecidable[4]. This motivated the SLTL^x-based approach, as did many others, to bound the problem space to reach decidability.

Let us consider approaches that use transition systems with data and some form of quantification in the specification language, such as [15, 23, 24, 30]. The decidability of verification and synthesis in such settings is usually obtained by imposing some kind of boundedness assumption on the domains of states in the transition systems. In comparison, the SLTL^x-based approach does not start with the bounded domain assumption, but boundedness is a consequence of the shape of transition systems corresponding to workflows, as they are acyclic. The complexity of the problem is also lower as a result.

The two approaches introduced by Gulwani, Jha et al. [52, 69] aim to synthesise finite *loop-free programs* from libraries of atomic program statements. The approaches restrict the number of resources that are available, as well as the structure of solutions to loop-free data flow diagrams. The loop-free program *synthesis from component libraries* problems can be seen as a scientific workflow synthesis.

[4] Chapter 3 shows that the unbounded SLTL^x synthesis problem is undecidable as well.

Instead of looking at scientific workflows as compositions of existing tools and operations, they can be seen as compositions of elements from component libraries. The two approaches [52, 69] use libraries of atomic program statements for synthesis, and therefore, they provide executable scripts as solutions. They both implement constraint-based synthesis from components using a satisfiability modulo theories (SMT) solver. The difference between the two approaches is the format in which the specification is provided. While [52] relies on a formal specification in first-order logic, [69] models it as an input-output oracle.

The approach introduced by Iannopollo et al. [61] provides another *synthesis from component libraries* approach. It sets a bound on the number of chosen components to solve the decidability problem. The approach instantiates the CEGIS (Counterexample-Guided Inductive Synthesis) paradigm, in which synthesis is carried out by an iterative algorithm. The algorithm comprises two steps. First, a discrete problem is solved by a constraint solver to retrieve a candidate solution. Second, the solution is verified according to a provided specification, and either accepted or added as a counterexample to the solver in the first step. The specification is given as Linear Temporal Logic (LTL)-based Assume/Guarantee (A/G) Contracts. An A/G contract describes the *assumptions* that a component makes on its environment and the *guarantees* it provides. The LTL A/G contract framework captures both, Assumption and Guarantee as Linear Temporal Logic (LTL) formulas. The components are annotated according to the inputs, outputs, assumptions and guarantees. The *LTL A/G contract-based synthesis* implementation is provided by the PYCO [60] tool. Despite the fact that the LTL A/G contract-based and the SLTL^x-based synthesis focus on different paradigms, they share many similarities[5]. The components under LTL A/G contracts can be seen as operations over inputs and outputs under SLTL^x, combined with some additional SLTL^x constraints which ensure A/G rules. In addition, both approaches provide data types in form of taxonomies, the difference is that the LTL A/G contract-based approach restricts them to trees.

[5]The theoretical framework allows for the components and system specifications to be more general than the A/G contracts. However, we focus on the LTL A/G contracts, as the corresponding implementation PYCO is provided.

CHAPTER 4

Workflow Synthesis as a SAT Problem

Abstract - Workflow synthesis is used by scientists to aid their exploration of possible solutions to their problems. As such, it is expected to be responsive and to be able to generate suggestions in a relatively short time frame. Considering the impressive progress of SAT solving in recent decades, we utilise SAT solving in our synthesis approach. This chapter presents semantic modelling used in the bounded SLTL^x synthesis problem, introduced in the previous chapter, and its translation into a propositional encoding, that can be fed to an off-the-shelf SAT solver. First, the chapter describes the modelling framework used to express user intent. It comprises the structure and format of the domain knowledge, namely, tool annotations and data taxonomies used to specify the problem specifications as SLTL^x constraints. Second, the chapter introduces a mechanism that captures the expected structure of the solution and the domain knowledge in propositional logic. Finally, the chapter presents a mechanism that translates arbitrary problem specifications, in the form of SLTL^x constraints, into propositional logic, and an illustrative synthesis run over such an encoding.
Several applications of APE in scientific case studies have shown that it is able to efficiently synthesise purposeful workflows. The case studies comprise research done within the scope of this dissertation (presented in detail in Chapter 6), as well as independent studies as part of the related work (described in Section 7.2). We use the example from the geovisualisation application domain as a running example in this chapter.

This chapter is based on the following publications:

Kasalica, V. & Lamprecht, A.-L., "Workflow Discovery with Semantic Constraints: The SAT-Based Implementation of APE", *Electronic Communications of the EASST*, vol. 78, May 2020, DOI: 10.14279/tuj.eceasst.78.1092, URL: https://journal.ub.tu-berlin.de/eceasst/article/view/1092 (visited on 05/17/2020).

Kasalica, V., Alechina, N., Lamprecht, A.-L. & Logan, B., "Instance-Aware Synthesis of Workflows Specified in Temporal Logic", *Journal of Artificial Intelligence Research (JAIR)*, 2023, Submitted and under review.

To allow for efficient use of the workflow composition in practice, this chapter focuses on a framework that solves the SLTL^x-based bounded synthesis problem (described in Chapter 3). We have shown that in practice, bounding the size of the solution workflows does not restrict the usability of the framework [76, 84]. In addition, the bounding of the search space makes the workflow synthesis problem NP-complete [72]. The complexity allows the usage of off-the-shelf solvers while keeping the full expressive power of the SLTL^x language. Considering the impressive progress of SAT solving [99] in recent decades, driven by annual competitions and impressive breakthroughs in the development of heuristics, we aim to utilise SAT solving in the synthesis approach.

To encode the problem in propositional logic (more specifically in conjunctive normal form, used by SAT solvers), we (1) must define a format in which the user can specify their intent, and (2) propose a mechanism for translating such model into propositional encoding.

4.1 Modelling User Intent

Chapter 1 describes obtaining an accurate description of a *user intent*, that is, the specification of the desired program, as one of the main challenges in program synthesis -.The main goals of modelling a user intent are an intuitive method of providing the specification and removing ambiguities in the specification.

To encode our problem as a constraint-solving problem, we need a strict model of the problem specifications. We follow the modelling approach introduced by Steffen et al. [41, 103, 129, 130] for this purpose. The approach has proven to be easy to use and effective in practice [3, 73, 111]. The modelling framework comprises two main components, (1) the *domain model*, defining the vocabulary of the domain (in the form of taxonomies) and input/output annotations (with respect to the given vocabulary) of the available operations, and (2) the *problem specification*, user-defined constraints that describe the given problem.

4.1.1 Modelling Domain Knowledge: Taxonomies

To properly capture user intent, it is essential to use clear and precise terminology. It should be abstract enough not to require users to be familiar with some low-level concepts (e.g., concrete tools) in the domain, and still concrete enough to accurately capture the desired goal. To support different levels of abstraction, terms for domain tools and data types are structured as *taxonomies*, tree-like structures composed of semantic *tools* and *data types*, respectively.

According to the aforementioned modelling framework a taxonomy $\mathcal{T} = (C, A, \rightarrow)$ is a weakly connected directed acyclic graph, where C represents a set of concrete elements from the domain (e.g., a concrete tool), A represents a set of conceptual elements, or classes, used to provide abstraction over the concrete elements, and $\rightarrow$ is a relation *is_a* (subsumption relation) over the two sets, specifically, over concrete and conceptual elements $c \rightarrow a$, where $c \in C, a \in A$, or pairs of conceptual elements $a_1 \rightarrow a_2$, where $a_1, a_2 \in A$.

The definition is based on the assumption that the subclass relation is jointly ex-

haustive, and leaf classes are mutually disjoint. Tool taxonomies support such an assumption, as executable operations (tools) used in a workflow implementation are atomic *low-level concepts* and are modelled as leaves of the taxonomy. Furthermore, each of the leaves represents a different program execution, and thus the leaves are mutually disjoint. However, this is not always the case.

Unlike executable operations, which are usually well defined (e.g., a concrete command line call, an API call, etc.), data types are more ambiguous, and require some post-processing to fit our assumption. Their definition might vary within different domains, and it is not possible to simply list all atomic data types. For example, the *HTML* format in a documentation building domain could be considered an atomic concept and would be modelled as a leaf of the taxonomy. However, in bioinformatics, more specifically in EDAM ontology that classifies concept in bioinformatics [64], the *HTML* format has multiple subclasses, one which is *FASTA-HTML* - a specialised HTML file format (for the simplicity of the following example let us assume that *FASTA-HTML* is the only subclass of the *HTML* format in the domain). Therefore, if we assume that the subclass relation is jointly exhaustive, there should be a *"plain HTML"* format that captures the non-*FASTA-HTML* files. Unfortunately in practice, the domains do not model such "plain" data types, as their semantics can be sometimes ambiguous. For example, in case of adding a new sub-format - *EMBL-HTML*, the semantics of *"plain HTML"* changes to non-*FASTA-HTML* and non-*EMBL-HTML*. As the models evolve, these concepts would change their scope as well.

Lack of "plain" types, unfortunately, in practice leads to some inaccurate annotations, as the umbrella terms (e.g., *HTML*) get used in place of "plain" (e.g., *"plain HTML"*) terms. For example, we notice that bio.tools [66], a community-curated repository of tool annotations in bioinformatics, comprises substantial amount of tool that work with *reports* in *"plain HTML"* format, which are annotated as *reports* in *HTML*[1]. This type of annotation is informative enough for an average user of the bio.tools platform. The main goal of the platform is "*to provide a comprehensive registry of software and databases, facilitating researchers from across the spectrum of biological and biomedical science to find, understand, utilise and cite the resources they need in their day-to-day work.*". However, when used in workflow synthesis [76], such annotations influence greatly the quality of our synthesis results, as they allow usage of specialised terms (e.g., *FASTA-HTML*) in place of the plain (e.g., *"plain HTML"*) ones. Therefore, we post-process such domains, introduce the "plain" types when needed, and adjust the tool annotations accordingly. These steps improve the quality of the domain models that we work with, and allow us to assume an exhaustive subsumption relation in the process.

Domain models, such as the bioinformatics domain we used in the previous example, show us that a single taxonomy term is often not sufficient to characterise a data object. In many domains, data are classified according to multiple disjoint criteria. We refer to them as *data dimensions* and each dimension represents a taxonomy where leaf classes are mutually disjoint.

[1]The list of tools is available at `https://bio.tools/t?sort=score&ord=desc&outputDataTypeID="data_2048"&outputDataFormatID="format_2331"`.

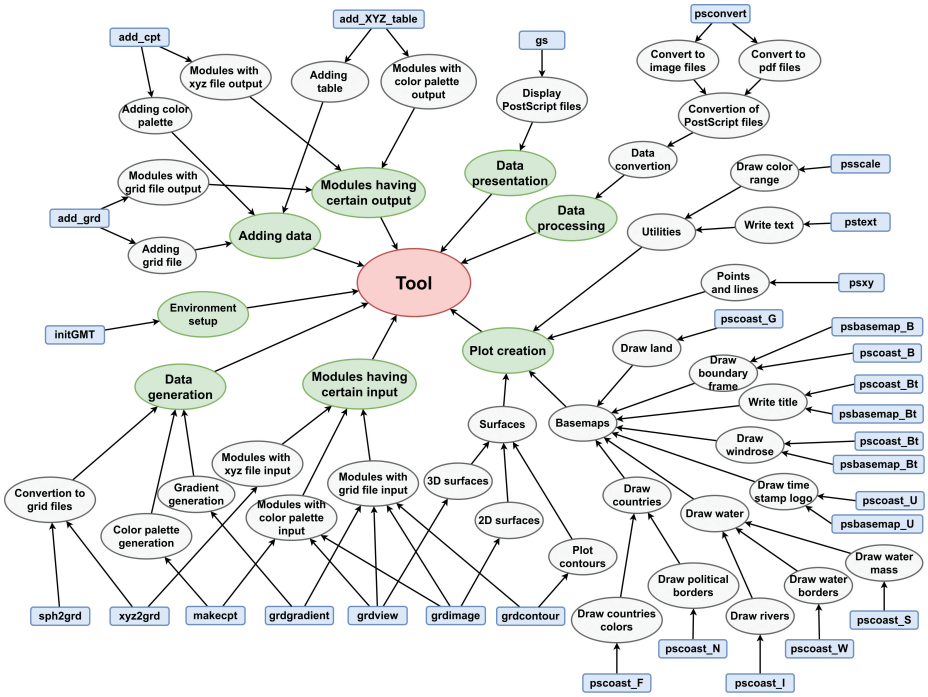


Figure 4.1: Tool Taxonomy for the Geovisualization Use Case Scenario

Data dimensions

In some domain models which are defined within a specific framework, such as the domain model comprising the Generic Mapping Tools (GMT) [141] in [73], one term is sufficient to characterise a data instance. For example, the content of a *PostScript file* in that domain can only be a plot. However, if we move to a broader domain of image manipulation, a *text file* can contain a colour code (e.g., *#F22B00*) that should be used, the content that should be printed, or even a type of font that should be used. In such a domain *textual file* would be the **format** of the data, while *font name* could be the **type** of the data. The such two-dimensional characterisation is quite common in the bioinformatics domain, provided by the EDAM ontology classification [64]. Furthermore, in a GIS question answering domain [84] a single geo-analytical concept (data instance) is characterised by four different aspects, (1) *geometric layer types*, which generalise geometric properties of layers, (2) *core concepts* of spatial information [85], which capture what these layers represent. (3) *measurement levels* of attributes, as well as the notion of (4) *extensiveness*.

To allow for accurate domain modelling in these domains, we introduce *data dimensions*, where each dimension characterises an aspect of a data instance. Within the pesented framework, each domain can have an arbitrary number d of data dimensions, and d–tuples are used to characterise data instances in the domain. To keep the domain modelling structured and coherent, the data dimensions are disjoint, and each is represented as a (sub-)taxonomy.

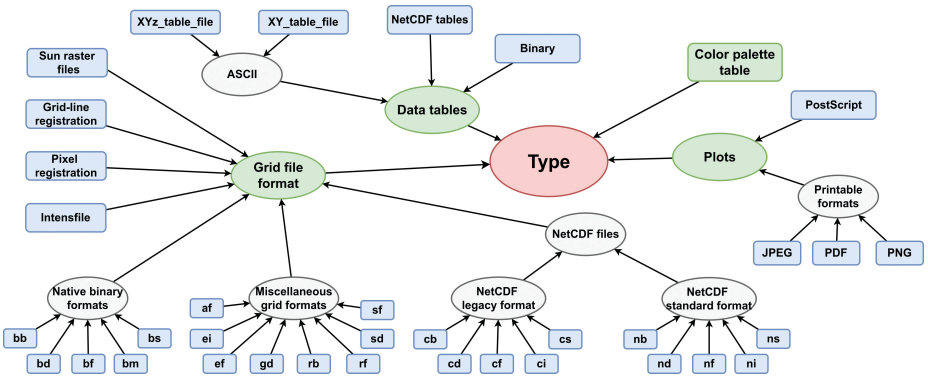


Figure 4.2: Type Taxonomy for the Geovisualization Use Case Scenario

Although data dimensions are not an explicit construct within SLTL^x, the syntax of the language allows conjunctions of data properties. Therefore, the difference between a one-dimensional and multi-dimensional domain reflects in specifying a property of a data instance, or conjunction of properties. For simplicity of the encoding, the following sections of the chapter focuses on a simple, one-dimensional type taxonomy.

Technically, we use a subset of the W3C Web Ontology Language (OWL) [9] to represent the taxonomies. OWL is a well-known Semantic Web language designed to represent ontologies, which has become the de facto standard for ontologies in many domains. Major domain ontologies, such as the EDAM data and methods ontology in bioinformatics, are provided in OWL and can thus directly be used in the presented framework. OWL can be used to describe complex relations between classes, but this framework only uses the concepts and concept inclusions (i.e., only the taxonomy part of the OWL file) to define and classify the taxonomy elements.

4.1.2 Modelling Domain Knowledge: Tool Annotations

To combine concrete tools, the synthesiser needs to know how these tools operate over data types. In practice, tools can perform complex operations over data types and implement various transformations on them. However, at the semantic level, we abstract these relations into two basic functions, known from the data-flow analysis:

- $use(\cdot) : C_M \to \mathcal{P}(C_T)$ – for a tool x to be executed, elements of the set of types $use(x)$ must be available
- $gen(\cdot) : C_M \to \mathcal{P}(C_T)$ – after execution of a tool x, elements of the set of type $gen(x)$ are available

The two functions can also be referred to as $input(\cdot)$ and $output(\cdot)$ respectively, which is more natural terminology for computational tools. Table 4.1 lists a selection of concrete tools from the geovisualization case study, each with its name, function description and its (possibly empty) sets of input and output types[2]. In

[2]Notice that the domain is one-dimensional, and thus, each input/output is annotated with one term. In the case of an n-dimensional domain, each input/output would be annotated using n-tuples.

practice, it has proven to be reasonable to only use the "payload" inputs/outputs in the annotation, and not all parameters that a tool might have. The latter tends to blow up the search space without actually being helpful to find new meaningful solutions. Therefore, in practice, each tool annotation represents a parametrised version of a software, i.e., a concrete tool instance where the parameters are fixed.

We distinguish two types of parameters. The first group comprises parameters that define the operation, e.g., a parameter that defines whether the GMT tool *pscoast*[3] plots water mass or political borders. Such parameters are crucial for defining the input/output dependencies and in practice often result in multiple input/output annotations per single software [73, 120]. The second group comprises parameters that calibrate the operation, e.g., the GMT parameter "+*w*"[4] can define the thickness of lines that depicts political borders. Such parameters are often set to a default value per tool annotation and can be manually adjusted before the execution when needed [73].

Name	Description	Type in	Type out
add_grd	Provide a grid file		NetCDF
grdgradient	Compute directional gradient	NetCDF	Intensfile
makecpt	Make color palette tables	cpt_file	cpt_file
...			
grdview	3D imaging of gridded data	NetCDF, cpt_file	PostScript
pscoast_W	Draw water borders	PostScript	PostScript
initGMT	Set-up the GMT environment		PostScript
gs	Display graphical files	Plots	

Table 4.1: Annotation of concrete tools in Geovisualization Use Case Scenario

Technically, we use a JSON representation for the tool annotations that follows the structure of the bio.tools schema [63] applied in the bio.tools registry [66]. For each tool function, we annotate its name and ID, the operation(s) that it performs, a set of inputs, a set of outputs and a command that corresponds to the tool execution. The last information is crucial for the automated implementation of the workflow. The current version of APE supports simple shell commands as well as CWL (Common Workflow Language) [8] annotations. The two result in a shell script or CWL workflow implementation, respectively.

4.1.3 Modelling Problem Specifications

The presented framework captures the problem specifications in SLTL^x (see detailed syntax in Chapter 3). The specification comprises (1) the initial data provided by the user, modelled as the output of the 0-th operation, (2) the data expected as the output, modelled as inputs to the $(n+1)$-th operation (where n is the length of the workflow), and (3) a set of constraints specified in SLTL^x that describe the desired workflow. The constraints can be specified directly in SLTL^x, or in a natural

[3] `https://docs.generic-mapping-tools.org/6.3/pscoast.html`
[4] `https://docs.generic-mapping-tools.org/6.3/cookbook/features.html#wpen-attrib`

language, by filling the natural language (NL) templates, that translate directly to SLTL^x. The solutions of the corresponding bounded workflow synthesis problem represent workflows in the given domain.

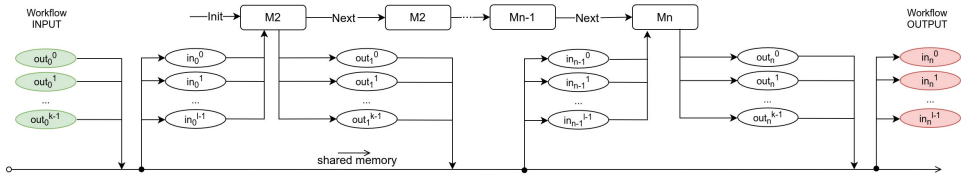


Figure 4.3: Shared memory design

4.2 Encoding Workflow Synthesis in Propositional Logic

To solve the synthesis problem using existing constraint solvers we incorporate some well-known ideas from planning as satisfiability [77, 78]. We provide an encoding of the general workflow structure, which is further enhanced with propositional constraints that correspond to the domain model, as well as the propositional encoding of the SLTL^x specified user intent. Finally, an off-the-shelf SAT solver (MiniSAT [38]) is used as a reasoning engine. This section covers each of the mentioned steps. First, we describe the encoding of the general workflow structure in propositional logic. Second, we discuss the encoding of the domain model, i.e., the encoding of the taxonomy structure and operation input and output dependencies. Third, we explain the propositional encoding of the temporal SLTL^x constraints that correspond to a user intent. Finally, we present the constraint solving and the synthesised solutions.

4.2.1 Encoding the Workflow Structure

As mentioned in the previous chapters, this dissertation focuses on computational pipelines, that is, linear workflows that represent the sequential execution of tools with no explicit branching on the control-flow level. These correspond to SLTL^x models, described in Chapter 3. To encode such structures, the presented framework utilises the *shared memory* design, i.e., workflow structure design where each tool can access the data created by any of the preceding tools. The design follows the structure presented in Figure 4.3[5].

In the same way that the SLTL^x models do not contain loops, we assume that each loop within a workflow can be flattened into a repetitive sequence of tools. Therefore, the framework is restricted to finite sequences of states and does not support the infinite behaviour of the system.

For our initial encoding, we need a formula which enforces that the crucial aspects of the workflow structure are encoded. Let L^o be the set of operations and L^t the

[5]Notice that the presented structures of the program resemble the transducer orchestration presented in Chapter 3. Each tool has a list of inputs and outputs which will be in turn connected.

set of all the data types (including the 'empty type' ϵ[6]), k and l the biggest input and output type indexes among the domain tools, respectively, i.e., the biggest number of inputs/outputs per tool, and

- $op(m_i)$ a unary predicate depicting that the operation *op* is implemented as the i-th *tool* in the sequence,
- $ty(in_i^j)$ a unary predicate depicts that the data type *ty* is used as the j-th *input* of the i-th tool in the sequence,
- $ty(out_i^j)$ a unary predicate depicts that the data type *ty* is generated as the j-th *output* of the i-th tool in the sequence,
- $Bind(in_{i1}^{j1}, out_{i2}^{j2})$ a binary predicate that *binds* tool *inputs* and the corresponding tool *outputs*, the example depicts that the tool input in_{i1}^{j1} is provided as the tool output out_{i2}^{j2}).
- $R(X, Y)$ a binary predicate depicting the *ancestor relation* R, as described in Chapter 3. The given example ensures that the data instance X is an ancestor of the data instance Y, where X and Y are tool inputs/outputs (e.g., in_i^j or out_i^j).

Under the given syntax, workflows of length n ($n \in N$), where n is a bound of our workflow, are encoded as follows:

$$
\begin{aligned}
[\![W]\!]_n := & \bigwedge_{i=1}^{n} \Big(\bigvee_{op \in L^o} op\,(m_i) \Big) \\
& \bigwedge_{i=0}^{n} \bigwedge_{j=0}^{k-1} \Big(\bigvee_{ty \in L^t} ty\,(in_i^j) \Big) \\
& \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{l-1} \Big(\bigvee_{ty \in L^t} ty\,(out_i^j) \Big) \\
& \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{k-1} \Big(\epsilon\,(in_i^j) \vee \bigvee_{p=0}^{i-1} \bigvee_{q=0}^{l-1} Bind(in_i^j, out_p^q) \Big)
\end{aligned}
\tag{4.1}
$$

An alternative representation of the formula that we use in this chapter is:

$$
\begin{aligned}
[\![W]\!]_n := & (\forall \mathbf{i} \in [1, n], \exists \mathbf{op} \in L^o)\ op\,(m_i) \wedge \\
& (\forall \mathbf{i} \in [0, n], \forall \mathbf{j} \in [0, k-1], \exists \mathbf{ty} \in L^t)\ ty\,(in_i^j) \wedge \\
& (\forall \mathbf{i} \in [0, n], \forall \mathbf{j} \in [0, k-1], \exists \mathbf{ty} \in L^t) ty\,(out_i^j) \wedge \\
& (\forall \mathbf{i} \in [0, n], \forall \mathbf{j} \in [0, k-1]) \\
& \quad \Big(\epsilon\,(in_i^j) \vee (\exists \mathbf{p} \in [0, i-1], \exists \mathbf{q} \in [0, l-1])\ Bind(in_i^j, out_p^q) \Big)
\end{aligned}
\tag{4.2}
$$

The formula ensures that each step of the workflow has an operation associated with it, and that each data instance (workflow inputs and outputs, tool inputs, tool outputs) has a type associated with it. Furthermore, it ensures that each data type

[6] ϵ represents the absence of data types, e.g., an output out_i^j labelled as $\epsilon(out_i^j)$ depicts that the i-th tool does not have the j-th output.

used as input is bound to an existing data type (available as a tool output or as one of the provided inputs). Finally, to ensure that the data binding relates pairs of the same data types, we extend the encoding with an additional formula, as follows.

$$\begin{aligned}[[W_{Bind}]]_n :=^{1)} & (\forall \mathbf{i} \in [0,n], \forall \mathbf{j} \in [0,k-1], \forall \mathbf{p} \in [0,i-1], \forall \mathbf{q} \in [0,l-1], \forall \mathbf{ty} \in L^t) \\ & Bind(in_i^j, out_p^q) \Rightarrow (ty\,(in_i^j) \Leftrightarrow ty\,(out_p^q)) \wedge \\ ^{2)} & (\forall \mathbf{i} \in [0,n], \forall \mathbf{j} \in [0,k-1], \forall \mathbf{p} \in [i,n], \forall \mathbf{q} \in [0,l-1])\ \neg Bind(in_i^j, out_p^q)\end{aligned} \tag{4.3}$$

The formula ensures that (1) each binding pair is annotated with the same data types, i.e., the type of the data instance in memory (created as a tool output) corresponds to the expected tool input type. In addition, the formula ensures that (2) the tool inputs can only reference data instances available in memory, i.e., a tool input cannot be a data instance that is not created yet.

Encoding of predicates used to describe additional information, not crucial in interpreting the structure of the actual workflow, such as the ancestor relation R, is presented separately in the following sections.

Example 1. *Our goal is to synthesise a workflow of length $n = 2$, where the biggest domain input and output type indexes are $k = 2$ and $l = 1$. The first step of the encoding is the workflow structure, as follows:*

$$\begin{aligned}[[W]]_2 := & ((\exists op_1, op_2 \in L^O)\ op_1(m_1) \wedge op_2(m_2)) \wedge \\ & ((\exists ty_1, ty_2, ty_3, ty_4, ty_5, ty_6 \in L^T)\ ty_1(in_0^0) \wedge ty_2(in_0^1) \wedge ty_3(in_1^0) \\ & \quad \wedge ty_4(in_1^1) \wedge ty_5(in_2^0) \wedge ty_6(in_2^1)) \wedge \\ & ((\exists ty_1, ty_2, ty_3 \in L^T)\ ty_1(out_0^0) \wedge ty_2(out_1^0) \wedge ty_3(out_2^0)) \wedge \\ & (\epsilon(in_0^0) \vee Bind(in_0^0, out_0^0)) \wedge \\ & (\epsilon(in_0^1) \vee Bind(in_0^1, out_0^0)) \wedge \\ & (\epsilon(in_1^0) \vee Bind(in_1^0, out_0^0) \vee Bind(in_1^0, out_1^0)) \wedge \\ & (\epsilon(in_1^1) \vee Bind(in_1^1, out_0^0) \vee Bind(in_1^1, out_1^0)) \wedge \\ & (\epsilon(in_2^0) \vee Bind(in_2^0, out_0^0) \vee Bind(in_2^0, out_1^0) \vee Bind(in_2^0, out_2^0)) \wedge \\ & (\epsilon(in_1^1) \vee Bind(in_2^1, out_0^0) \vee Bind(in_2^1, out_1^0) \vee Bind(in_2^1, out_2^0))\end{aligned}$$

An additional formula is introduced to ensure the data binding. Segments of the formula are encoded as follows:

$$
\begin{aligned}
[[W_{Bind}]]_2 \coloneqq & \big(\neg Bind(in_0^0, out_0^0) \vee ((\forall ty \in L^T) ty(in_0^0) \Leftrightarrow ty(out_0^0)\big) \wedge \\
& \big(\neg Bind(in_0^1, out_0^0) \vee ((\forall ty \in L^T) ty(in_0^1) \Leftrightarrow ty(out_0^0)\big) \wedge ... \wedge \\
& \big(\neg Bind(in_2^1, out_1^0) \vee ((\forall ty \in L^T) ty(in_2^1) \Leftrightarrow ty(out_1^0)\big) \wedge \\
& \big(\neg Bind(in_2^1, out_2^0) \vee ((\forall ty \in L^T) ty(in_2^1) \Leftrightarrow ty(out_2^0)\big) \wedge \\
& \neg Bind(in_0^0, out_1^0) \wedge \neg Bind(in_0^0, out_2^0) \wedge \neg Bind(in_0^1, out_1^0) \wedge \\
& \neg Bind(in_0^1, out_2^0) \wedge \neg Bind(in_1^0, out_2^0) \wedge \neg Bind(in_1^1, out_2^0)
\end{aligned}
$$

4.3 Encoding the Domain Model

Once the initial structure has been encoded, we define the rules that translate our domain knowledge into a set of propositional formulas. This includes (1) preserving input and output types for each tool, (2) preserving the classifications defined by the taxonomy, and (3) ensuring that none of the states in our encoded transition system violates the intended structure, that is, ensuring that each state that corresponds to a tool (or type) is represented by exactly one tool (or type) predicate. These constraints ensure that our structure can be unambiguously mapped to exactly one workflow representation. The rest of this section presents this encoding of the domain model.

Preserving tool inputs. Let n be the workflow bound and k the biggest input type index. To preserve tool input relation, for each tool X and the list of types $Y_1, Y_2, ..., Y_p$, where $p \leq k$ and $Y_j \in input(X)$ for $j \in [1, p]$, we define the formula:

$$
\begin{aligned}
[[In(X)]]_n \coloneqq & (\forall i \in [1, n]) \\
& \Big(X(m_i) \Rightarrow \big((\forall j \in [0, p-1])\ Y_j(in_i^j) \wedge (\forall j \in [p, k])\ \epsilon(in_i^j)\big)\Big) \qquad (4.4)
\end{aligned}
$$

The formula encodes a condition where the usage of the tool X in a certain tool state m_i requires precisely the types $Y_1, ..., Y_p$ to be provided as inputs to it, i.e., $Y_j(in_i^j)$ where $j \in [0, p]$. Notice the order of inputs is fixed, according to the tool annotations provided.

Example 2. *We extend Example 1 with the encoding of tool inputs. Remember that the length of the workflow is $n = 2$ and input index is $k = 2$. To simplify the example we present the encoding of a single tool input, using the tool makecpt (see Table 4.1), which makes a 'color palette tables' and the input type it requires is a 'cpt_file'. The encoding is as follows:*

$$
\begin{aligned}
[[In(makecpt)]]_2 \coloneqq & \big(makecpt(m_0) \Rightarrow cpt\_file(in_0^0) \wedge \epsilon(in_0^1)\big) \wedge \\
& \big(makecpt(m_1) \Rightarrow cpt\_file(in_1^0) \wedge \epsilon(in_1^1)\big) \wedge \\
& \big(makecpt(m_2) \Rightarrow cpt\_file(in_2^0) \wedge \epsilon(in_2^1)\big)
\end{aligned}
$$

The second output state is labelled as an empty type as the input index of the domain (number of input states) is higher than the number of expected inputs by the *makecpt* operation.

Preserving tool outputs. The following set of formulas encodes the preservation of the tool output relations. Let n be the workflow bound and l the biggest output type index. For each tool X and list of types $Y_1, .., Y_p$, where $p \leq l$ and $Y_j \in output(X)$, $\forall j \in [1, p]$ we define the formula:

$$[[Out(X)]]_n := (\forall i \in [1, n]) \Big(X(m_i) \Rightarrow \big((\forall j \in [0, p-1])\ Y_j(out_i^j) \wedge (\forall j \in [p, l])\ \epsilon(out_i^j) \big) \Big) \quad (4.5)$$

1 The formula encodes a condition where the usage of the tool X in a tool state m_i, enforces strictly the types $Y_1, .., Y_p$ to be provided in the output type states that follow the operation, i.e., $Y_j(out_i^j)$, for $j \in [0, p]$. In case p is smaller than the number of output states, the rest of the type states are empty.

Example 3. *We extend Example 2 with the encoding of tool outputs. Remember that the length of the workflow is $n = 2$ and the output index is $l = 1$. To simplify the example we show the encoding of a single tool input for the mentioned tool makecpt (see Table 4.1). The tool has one output type, which is cpt_file. The encoding is as follows:*

$$\begin{aligned} [[Out(makecpt)]]_2 := & \big(makecpt(m_0) \Rightarrow cpt\_file(out_0^0) \big) \wedge \\ & \big(makecpt(m_1) \Rightarrow cpt\_file(out_1^0) \big) \wedge \\ & \big(makecpt(m_2) \Rightarrow cpt\_file(out_2^0) \big) \end{aligned}$$

Notice that none of the output states was labelled with an empty type as the number of outputs of the *makecpt* operation is the same as the output index of the domain, i.e., the number of output states in the model.

Preserving taxonomy classification. To preserve a classification provided by the taxonomy, we introduce a set of formulas that encode the dependency between tool/type ontology concepts and their subclasses. At the encoding step, we assume that the taxonomies implement an exhaustive subsumption relation where all leaf concepts are mutually disjoint. As we have mentioned in Section 4.1.1, tool taxonomies adhere to the assumption, while the type taxonomies usually require a post-processing step.

Introducing "plain" classes improves the quality of the domain annotations and facilitates exhaustive subsumption relations. Furthermore, we combine this approach with the concept of multi-dimensional data (described in Section 4.1.1), which ensures that leaves within the same dimension are mutually disjoint.

Exhaustive subsumption relations imply that once a non-leaf tool/type has been used in a state, at least one of its subclasses needs to be used as well, and vice versa. For each non-leaf taxonomy term X in the tool taxonomy and the list of its subclasses $Y_1, ..Y_p$, such that $Y_i \rightarrow X, \forall i \in [1, p]$ we define the following formula:

$$\begin{aligned} [[Tax^{op}(X)]]_n := (\forall i \in [1, n]) \big(& (X(m_i) \Rightarrow (\exists j \in [1, p])\ Y_j(m_i)) \\ & \wedge (\forall j \in [1, p])(Y_j(m_i) \Rightarrow X(m_i)) \big) \end{aligned} \quad (4.6)$$

The first part of the formula enforces the usage of at least one of the sub-tools of X in a certain state, providing that X was used in that state as well. The second

part of the formula enforces usage of the tool X, providing that at least one of its sub-tools is used in the same state.

Similarly, for each non-leaf taxonomy term X in the type taxonomy and the list of its subclasses $Y_1, ..Y_p$, such that $Y_i \rightarrow X, \forall i \in [1,p]$ where k and l are the biggest input and output type indexes among the domain tools, we define the following formula:

$$\begin{aligned}[[Tax^t(X)]]_n :=& (\forall i \in [1,n], \forall p \in [0,k-1], \forall q \in [0,l-1]) \\ & \big(X(in_i^p) \Rightarrow (\exists j \in [1,p])\ Y_j(in_i^p) \wedge \\ & (\forall j \in [1,p])(Y_j(in_i^p) \Rightarrow X(in_i^p))\big) \wedge \\ & \big(X(out_j^q) \Rightarrow (\exists j \in [1,p])\ Y_j(out_j^q) \wedge \\ & (\forall j \in [1,p])(Y_j(out_j^q) \Rightarrow X(out_j^q))\big) \end{aligned} \tag{4.7}$$

The transition to the encoding of the type taxonomy classification is trivial, as the only difference is the state that is encoded (e.g., in_i^j instead of m_i), and thus, the complete encoding is omitted.

Example 4. *We extend Example 3 with the encoding of tool and type taxonomies. To keep the example simple, we show the encoding of a sub-taxonomy that consists of an abstract tool Write title and its two sub-tools pscoast_Bt and psbasemap_Bt (see the right-hand side of Figure 4.1). For simplicity of the notion the tools are abbreviated as WT, C_Bt and B_Bt, respectively. The encoding is as follows:*

$$\begin{aligned}[[Tax^{op}(WT)]]_2 :=& \big(WT(m_1) \Rightarrow C\_Bt(m_1) \vee B\_Bt(m_1)\big) \wedge \\ & \big(C\_Bt(m_1) \Rightarrow WT(m_1)\big) \wedge \big(B\_Bt(m_1) \Rightarrow WT(m_1)\big) \wedge \\ & \big(WT(m_2) \Rightarrow C\_Bt(m_2) \vee B\_Bt(m_2)\big) \wedge \\ & \big(C\_Bt(m_2) \Rightarrow WT(m_2)\big) \wedge \big(B\_Bt(m_2) \Rightarrow WT(m_2)\big) \wedge \end{aligned}$$

Enforcing mutual exclusion of concrete tools/types. To ensure that each solution provided by the solver corresponds to exactly one workflow structure, we have to avoid conflicts of using two different concrete tools or types in the same state of the structure. As we have previously mentioned, we assume an exhaustive subsumption relation that models concrete tools/types as taxonomy leaves, and thus, to enforce the tool/type uniqueness it is sufficient to preserve mutually disjoint taxonomy leaves. Let n be the workflow bound. For each pair of concrete tools X_1 and X_2, we introduce the formula

$$[[Conf(X_1, X_2)]]_n := (\forall i \in [1,n])(\neg X_1(m_i) \vee \neg X_2(m_i)) \tag{4.8}$$

to eliminate conflicts regarding the usage of multiple concrete tools simultaneously. The formula forbids the usage of two different concrete tools in a single tool state. Similarly, we can encode mutual exclusion of a pair of types Y_1 and Y_2, where k and l are the biggest input and output type indexes among the domain tools, as follows:

$$\begin{aligned}[[Conf(Y_1, Y_2)]]_n :=& (\forall i \in [0,n])\big((\forall j \in [0,k-1])(\neg Y_1(in_i^j) \vee \neg Y_2(in_i^j)) \\ & (\forall j \in [0,l-1])(\neg Y_1(out_i^j) \vee \neg Y_2(out_i^j))\big) \end{aligned} \tag{4.9}$$

Example 5. *We extend Example 4 with the encoding of mutual exclusion of concrete tools and types. To simplify the example we only show the encoding of mutual exclusion of two concrete tools, makecpt and gs (see Figure 4.1), while omitting the mutual exclusion of data types. The encoding is as follows:*

$$[\![Conf(makecpt, gs)]\!]_2 := \big(\neg makecpt(m_1) \vee \neg gs(m_1)\big) \wedge \big(\neg makecpt(m_2) \vee \neg gs(m_2)\big)$$

4.4 Encoding the Temporal Constraints

To restrict the synthesis to workflows that fit a specific task we use the SLTL^x specification. Users use the specification to define the provided workflow inputs, the desired workflow outputs, and to describe the expected workflow structure.

To accomplish that, we provide a mechanism for transforming SLTL^x formulas into propositional logic. The transformation framework is an extension of our previous work that transforms SLTL formulas to propositional logic [75]. We extend the syntax to cover SLTL^x, accompanied by a parser for the logic.

Both transformations, of the SLTL and the SLTL^x language, are based on the framework introduced by Biere et al. [17], which provides a mechanism for transforming arbitrary LTL formulas into propositional formulas. The paper distinguishes between transformations of LTL formulas that include loops in their path, and those that do not. As we are dealing with loop-free computational workflows here, we focus on the latter.

To encode the SLTL^x formulas in propositional logic, we introduce some additional propositional predicates.

The ancestor relation (R) is an SLTL^x concept introduced in Chapter 3 used to define a transitive relation between generated data objects and their 'ancestors', i.e., objects used to generate it. Let n be the workflow bound, k and l the biggest input and output indexes. We define the corresponding propositional encoding as follows.

$$\begin{aligned}[\![R(X)]\!]_n :=& (\forall i \in [1,n], \forall p \in [0,k-1], \forall q \in [0,l-1])\ \big(R(in_i^p, in_i^p) \wedge R(out_i^q, out_i^q) \wedge \\ & R(in_i^p, out_i^q) \wedge (\forall j \in [i,n])\ Bind(in_i^p, out_j^q) \Rightarrow R(out_j^q, in_i^p)\big) \wedge \\ & (\forall i_1, i_2, i_3 \in [1,n], \forall p_1, p_2, p_3 \in [0,l-1]) \\ & (R(out_{i_1}^{p_1}, out_{i_2}^{p_2}) \wedge R(out_{i_2}^{p_2}, out_{i_3}^{p_3}) \Rightarrow R(out_{i_1}^{p_1}, out_{i_3}^{p_3})) \end{aligned} \tag{4.10}$$

The identity relation (IS) is a predicate we introduce to skolemize our SLTL^x formula, with respect to the first-order logic concepts. Let n be the workflow bound, k and l the biggest input and output indexes, the predicates are encoded as follows.

$$[\![IS(X)]\!]_n := (\forall i \in [1,n], \forall p \in [0,l-1]) IS(out_i^p, out_i^p) \tag{4.11}$$

Translation of SLTL^x to the propositional encoding is presented in the following definition.

Definition 16. *Let n be the workflow bound, k and l the biggest input and output indexes, respectively, $t, t_1, t_2, ..$ are SLTL^x terms (constants or variables) and x a variable.*

The notion $[\![\Phi]\!]_n^i$ for $i \in [0,n]$ refers to the interpretation of the SLTL^x formula Φ in the i-th state of the path of length n, where states are described as sets of data instances and properties over them. States are connected by transitions labelled with operations. Translation of SLTL^x formulas into propositional format is defined as follows:

$$
\begin{aligned}
[\![true]\!]_n^i &:= true \\
[\![P(t)]\!]_n^i &:= P(t) \wedge \big((\forall p \in [0,i], \forall q \in [0,l-1]) \\
&\qquad IS(t, out_p^q) \Rightarrow P(out_p^q)\big) \\
[\![t_1 = t_2]\!]_n^i &:= (\forall p \in [0,i], \forall [0,l-1]) \\
&\qquad IS(t_1, out_p^q) <=> IS(t_2, out_p^q) \\
[\![R(t_1,t_2)]\!]_n^i &:= R(t_1,t_2) \ \wedge \big((\forall p_1,p_2 \in [0,i], \forall q_1,q_2 \in [0,l-1]) \\
&\qquad IS(i_1, out_{p_1}^{q_1}) \wedge IS(i_2, out_{p_2}^{q_2}) \Rightarrow R(out_{p_1}^{q_1}, out_{p_2}^{q_2})\big) \\
[\![\neg\Phi]\!]_n^i &:= \neg [\![\Phi]\!]_n^i \\
[\![\Phi_1 \wedge \Phi_2]\!]_n^i &:= [\![\Phi_1]\!]_n^i \wedge [\![\Phi_2]\!]_n^i \\
[\![\Phi_1 \vee \Phi_2]\!]_n^i &:= [\![\Phi_1]\!]_n^i \vee [\![\Phi_2]\!]_n^i \\
[\![(\exists x)\,\Phi]\!]_n^i &:= (\exists p \in [0,i], \exists q \in [0,l-1])\ IS(x, out_p^q) \wedge [\![\Phi]\!]_n^i \\
[\![(\forall x)\,\Phi]\!]_n^i &:= (\forall p \in [0,i], \forall q \in [0,l-1])\ IS(x, out_p^q) \wedge [\![\Phi]\!]_n^i \\
[\![\langle Op^{a,b}(t_1,\ldots,t_{a+b})\rangle\Phi]\!]_n^i &:= Op(m_i) \ \wedge [\![\boldsymbol{X}\Phi]\!]_n^i \wedge \\
&\qquad (\forall p \in [1,a], \exists q \in [0,k-1])\ IS(t_p, in_p^q) \ \wedge \\
&\qquad (\forall p \in [a+1,a+b], \exists q \in [0,l-1])\ IS(t_p, out_p^q) \\
[\![\boldsymbol{G}\Phi]\!]_n^i &:= [\![\Phi]\!]_n^i \wedge [\![\boldsymbol{G}\Phi]\!]_n^{i+1} \qquad \textit{where } [\![\boldsymbol{G}\Phi]\!]_n^n := [\![\Phi]\!]_n^n \\
[\![\boldsymbol{F}\Phi]\!]_n^i &:= [\![\Phi]\!]_n^i \vee [\![\boldsymbol{F}\Phi]\!]_n^{i+1} \qquad \textit{where } [\![\boldsymbol{F}\Phi]\!]_n^n := [\![\Phi]\!]_n^n \\
[\![\boldsymbol{X}\Phi]\!]_n^i &:= [\![\Phi]\!]_n^{i+1} \qquad \textit{where } [\![\boldsymbol{X}\Phi]\!]_n^n := false \\
[\![\Phi_1 \boldsymbol{U} \Phi_2]\!]_n^i &:= [\![\Phi_2]\!]_n^i \vee ([\![\Phi_1]\!]_n^i \wedge [\![\Phi_1 \boldsymbol{U} \Phi_2]\!]_n^{i+1})
\end{aligned}
$$

Base case:

$$[\![\Phi]\!]_n^n := false$$

The translation rules are used to transform the user specification from SLTL^x language into propositional logic. The following example illustrates an SLTL^x transformation.

Example 6. *We extend Example 5 with the encoding of the SLTL^x formula $\phi = \boldsymbol{G}\neg\langle grdview^{0,0}()\rangle true$ ("Do not use tool grdview in the solution."). The translation of ϕ to a propositional formula is as follows:*

$$
\begin{aligned}
&[\![\boldsymbol{G}\neg\langle grdview^{0,0}()\rangle true]\!]_2^0 := \neg grdview(m_1) \wedge [\![\boldsymbol{G}\neg\langle grdview^{0,0}()\rangle true]\!]_2^1, \quad \textit{where} \\
&[\![\boldsymbol{G}\neg\langle grdview^{0,0}()\rangle true]\!]_2^1 := \neg grdview(m_2) \wedge [\![\boldsymbol{G}\neg\langle grdview^{0,0}()\rangle true]\!]_2^2, \quad \textit{where} \\
&[\![\boldsymbol{G}\neg\langle grdview^{0,0}()\rangle true]\!]_2^2 := [\![\neg\langle grdview^{0,0}()\rangle true]\!]_2^2 := \neg false = true
\end{aligned}
$$

Apart from directly writing SLTL^x constraints to specify a user intent, we allow for usage of closed-text, i.e., natural language (NL), templates that correspond to SLTL^x formulas. For example, user can specify "Use data type *Ty* in the solution" instead of writing $\mathbf{F}(\exists x)Ty(x)$ (see Table 4.2 for the full list of templates). The motivation for such a feature is threefold:

1. Practice has shown that some types of constraints are used more frequently than others, and thus, to simplify the interaction with the framework, we provide NL templates for such constraints.
2. By providing frequently used templates in natural language, users are not required to learn the SLTL^x syntax to use the framework.
3. Having fixed templates allows us to optimise the implementation of the underlying SLTL^x formulas as shown in our recent work [75]. Improving the performance of the encoding when compared to the rules used in Definition 16.

By default, the templates encode the SLTL^x formula according to the translation presented in Definition 16. However, the encoding of commonly used formulas can be unnecessarily complex. One such example is the SLTL^x constraint that specifies the last tool in the workflow (see **T7** from Table 4.2). To express such a constraint in SLTL^x, e.g., "Use tool *Op* as the last tool in the solution", we have to use different modal operators. Some of the possible encoding approaches are presented as follows.

$$[\![T_7]\!]_n \coloneqq [\![\mathbf{F}\big(\langle Op^{0,0}()\rangle true \wedge \mathbf{X}\ false\big)\ ]\!]_n^0 \tag{4.12}$$

or

$$[\![T_7]\!]_n \coloneqq [\![\mathbf{F}\langle Op^{0,0}()\rangle true\ \wedge\ \mathbf{G}(\mathbf{X}\,\mathbf{X} true \vee \langle Op^{0,0}()\rangle\, true \vee \neg\mathbf{X} true)]\!]_n^0 \tag{4.13}$$

or

$$[\![T_7]\!]_n \coloneqq [\![\mathbf{G}\ \mathbf{F}\langle Op^{0,0}()\rangle true\ ]\!]_n^0 \tag{4.14}$$

Each of the SLTL^x formulas has a different structure that determines the complexity of the propositional encoding. For example, the nesting of the modal operators causes exponential blowup with respect to the size of the propositional encoding in conjunctive normal form (CNF)[7]. Therefore, some encoding structures are preferred over others. However, even the optimal CNF encoding generated from SLTL^x structure using Definition 16 can often be further simplified. For example, the presented type of constraints can be directly encoded in the workflow structure, as it has a fixed bound, optimising the encoding. Thus, this framework rewrites the encoding into a simpler formula, where n is the bound of the workflow:

$$[\![T_7]\!]_n \coloneqq Op(m_n) \tag{4.15}$$

Although the simplification in the encoding is not always as straightforward as the previous example, encoding of other NL templates can be optimised. For example, the constraint **TX** from Table 4.2 ("Use operation *Op* with an input of type *Ty*") can avoid the direct transformation from SLTL^x, which introduces a recursive call over all available data types at each step. The direct encoding of the template is as follows:

[7] SAT solvers expect encoding to be provided in CNF.

$$[[T_X]]_n := (\exists i \in [1,n], \exists y \in [0.p-1])(Op(m_i) \wedge Ty(in_i^j)) \tag{4.16}$$

On the other hand, there can be templates that are not difficult to encode, but difficult to specify. For example, the constraint **TX** from Table 4.2 ("Identical tools should not be connected via output/input", i.e., we should not have port binding between tools/transducers of the same type) is difficult to specify manually in SLTL^x. The user must specify the formula of the format:

$$\neg\exists x(\mathbf{F} < Op^{0,1}(x) > (\mathbf{F} < Op^{1,0}(x) > true)) \tag{4.17}$$

for each annotated tool *Op* individually[8]. Therefore, there should be *k* such constraints, where *k* is the number of distinct transducers/tool annotations in the domain. In this scenario, the corresponding NL template improves the encoding, but more importantly, it simplifies the specification step, even for the user that is familiar with SLTL^x.

The optimised encoding of the NL templates reflects in the improved translation time (from the SLTL^x the CNF encoding), as presented in Chapter 7. The runtime improvement ultimately depends on the complexity of the underlying SLTL^x formula. For example, simple formulas that utilise a single modal operator, such as "$\mathbf{F} < Op^{0,1}() > true$" show between 10% and 20% runtime improvement. On the other hand, complex constructs, that involve nesting of modal operators, show much bigger improvements, e.g., encoding could take a few seconds instead of more than an hour. The complete evaluation is presented in Section 7.1.

4.5 Solving the Encoded Problem

Once the complete encoding is provided, it is sent to the MiniSAT solver [38] to perform the synthesis. Based on the solutions provided by the solver, candidate workflows and their executable implementations are provided to the user. We perform the search for possible workflows until the first depths where solutions are found (the search depth is the same as the length of the solutions). Usually, the shortest solutions are also the most relevant with respect to the workflow specification, as they present the smallest number of steps necessary to satisfy it. To illustrate, if we look at our running example (see Examples 1 - 6) and assume that there is no initial input provided to the workflow, some of the proposed solutions would be[9]:

- initGMT → gs
- initGMT → pscoast_W
- add_grd → makecpt

where the arrows denote the order in which the tools are being executed, i.e., the first solution suggests using tools *initGMT* and *gs* in a sequence. For the current specification, each solution is of length 2 and none of the solutions includes the tool

[8]The operation *Op* cannot be set to an abstract class, as it would be too restrictive. For example, if we set *Op* to be a plotting operation, such as $\neg\exists x(\mathbf{F} < \textit{Plot creation}^{0,1}(x) > (\mathbf{F} < \textit{Plot creation}^{1,0}(x) > true))$, the constraints forbids using any two plotting operations in a sequence.

[9]We present simplified structure of the solutions, illustrating only the operations performed.

ID	Constraints in Natural Language	Constraints in SLTL
T1	If tool Op_1 is used, tool Op_2 has to be used subsequently	$G(\neg < {Op_1}^{0,0}() > true \vee XF < {Op_2}^{0,0}() > true)$
T2	If tool Op_1 is used, tool Op_2 cannot be used subsequently	$G(\neg < {Op_1}^{0,0}() > true \vee XG\neg < {Op_2}^{0,0}() > true)$
T3	If tool Op_2 is used, tool Op_1 must have been its direct predecessor in the sequence	$G(\neg\ X\ Op_2^0)true \vee (Op_1^{0,0})\ Op_2^{0,0})true))$
T4	If tool Op_1 is used, tool Op_2 has to be used next in the sequence	$G(\neg < {Op_1}^{0,0}() > true \vee X < {Op_2}^{0,0}() > true)$
T5	Use tool Op_1 in the solution	$F < Op_1^0, 0() > true$
T6	Do not use tool Op_1 in the solution	$G\neg < Op_1^0, 0() > true$
T7	Use Op_1 as last tool in the solution.	$F < {Op_1}^{0,0}() > true \wedge G(\neg < {Op_1}^{0,0}() > true \vee \neg X\ X\ true)$
T8	Use type Ty in the solution	$F\exists x Ty(x)$
T9	Do not use type Ty in the solution	$\neg F\exists x Ty(x)$
T10	Tool Op_1 should generate output used by operation Op_2	$F\exists x(< Op^{0,1}(x) > (F < Op^{1,0}(x) > true))$
T11	Tool Op_1 should never generate output used by operationOp_2	$\neg F\exists x(< Op^{0,1}(x) > (F < Op^{1,0}(x) > true))$
T12	Use tool Op with an input of type Ty	$F\exists x(Ty(x) \wedge < Op^{1,0}(x) > true))$
T13	Use tool Op to generate output of type Ty	$F\exists x(< Op^{0,1}(x) > Ty(x)))$
T14	Identical tools should not be connected via output/input	$\neg\exists x(F < Op^{0,1}(x) > (F < Op^{1,0}(x) > true))$

Table 4.2: User Intent: NL templates for SLTL^x formulas.

grdview, due to the user intent constraint that excludes this tool (see Example 6). Additionally, each tool that is suggested as first in the sequence does not require any input, considering that we did not provide any initial workflow input. Similarly, the second tool is limited to the tools that require no input or the input that was provided as the output of the first tool. To illustrate, we will elaborate on the first proposed solution. It represents a workflow that uses *initGMT* command to instantiate a GMT program and to generate an empty map, while the second command - *gs*, displays the generated map to the user. Although the workflow does not perform any notable computations, it is one of the smallest programs that satisfy the constraints presented in Examples 1 - 6. That is why an accurate specification of the program is as important as the program synthesis algorithm itself.

The example presents a trivial encoding of the problem, used to illustrate the translation mechanism. The original geovisualisation case study [73] includes a much larger set of constraints, such as "Use operation *2D_surfaces*" (in SLTL^x expressed as **F** < *2D_surfaces*$^{0,0}() > true$), "Use operation *Draw_political_borders*" (in SLTL^x expressed as **F** < *Draw_political_borders*$^{0,0}() > true$), etc. The complete example presented in the case study enforces the usage of nine specific operations and results in workflows of length 16. The solutions accurately solve the given problem. We present the full problem specification and the generated solutions in Chapter 6.

CHAPTER 5

APE (the Automated Pipeline Explorer) v2

Abstract - The chapter introduces APE v2 (the Automated Pipeline Explorer) as a command-line tool and Java API for the automated composition of scientific workflows. In addition, it describes APE Web, a web interface built on top of the APE API. The goal of the APE framework is to provide a robust and lightweight solution to existing synthesis problems in scientific domains.
For a domain set-up, APE requires a domain ontology and semantically annotated tools. The domain can then be utilised to synthesise scientific workflows based on an SLTL^x specification. APE v2 implements the transformation algorithm presented in Chapter 4 to encode the specification in propositional logic, and use the MiniSAT solver to synthesise the corresponding solutions.
The chapter also presents a mechanism used to transform generated APE workflows into the CWL (Common Workflow Language) format. Furthermore, the mechanism can be adapted to suit other workflow languages (e.g., NextFlow, SnakeMake).

This chapter is based on the following publications:

Kasalica, V. & Lamprecht, A.-L., "APE: A Command-Line Tool and API for Automated Workflow Composition", in: *Computational Science – ICCS 2020*, ed. by Krzhizhanovskaya, V. V., Závodszky, G., Lees, M. H., Dongarra, J. J., Sloot, P. M. A., Brissos, S., et al., Cham: Springer International Publishing, 2020, pp. 464–476, ISBN: 978-3-030-50436-6.

Kasalica, V., Alechina, N., Lamprecht, A.-L. & Logan, B., "Instance-Aware Synthesis of Workflows Specified in Temporal Logic", *Journal of Artificial Intelligence Research (JAIR)*, 2023, Submitted and under review.

This chapter introduces APE v2 [74] as the software that implements SLTL^x-based workflow synthesis (described in Chapter 3). The APE v2 framework uses the transformations presented in Chapter 4 to encode the SLTL^x-based bounded synthesis problem in propositional logic and use the MiniSAT [38] solver to synthesise the solution.

Section 5.1 presents the architecture of the APE v2 framework. Section 5.2 illustrates how to set-up APE for use by providing a semantic domain model. Section 5.3 focuses on the interfaces for the automated composition of workflows based on the domain model and custom workflow specifications. It presents the APE application programming interface (API), the command line interface (CLI) and the graphical browser-based interface (GUI) built on top of the APE API. Section 5.4 describes how APE-composed workflows can be further transformed into standard workflow formats. Finally, Section 5.5 presents some related tools that provide scientific workflow synthesis.

APE v2 is available as an open source project at `https://github.com/sanctuuary/APE`.

5.1 Architecture

The architecture of APE v2 is shown in Figure 5.1.

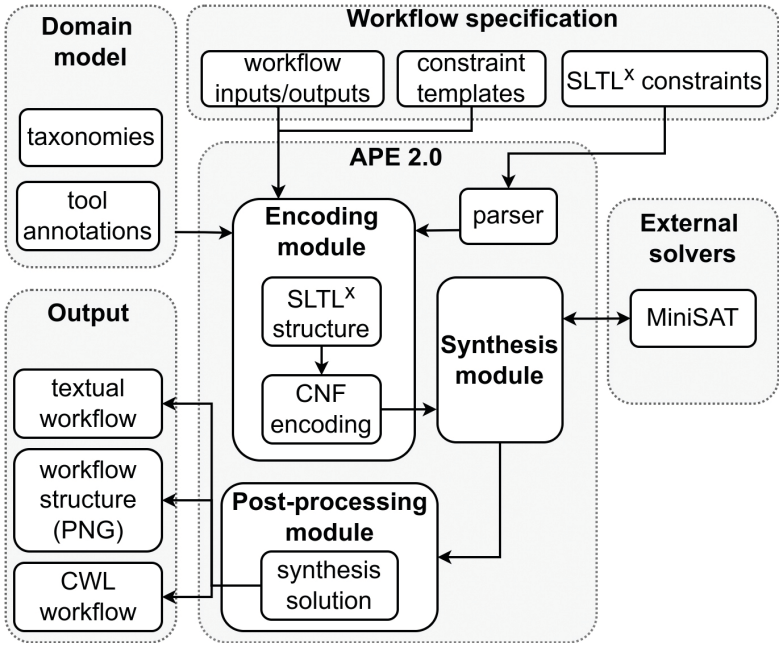


Figure 5.1: APE v2 architecture

The input to APE v2 is a domain model that configures the framework to the application domain, and a workflow specification describing the desired workflow, as

described in Chapter 4. The domain model, comprising taxonomies and semantic tool annotations, is typically provided by a domain expert. The workflow specification comes from an end user (e.g., a researcher) and consists of the available inputs, desired outputs and additional constraints.

APE v2 provides an interface for users to specify arbitrary SLTL^x formulas when describing the problem. Furthermore, the framework provides a set of most commonly used SLTL^x constraints in the form of natural language templates, elaborated in Section 4.4. Examples of templates available in APE v2 include "*Use operation X in the workflow*", formally $\boldsymbol{F}(\langle X^{0,0}() \rangle \text{ true})$, and "*Tool Y must not take data processed by tool X*", formally $\neg \boldsymbol{F}(\exists x \langle X^{0,1}(x) \rangle \boldsymbol{F}(\exists y \langle Y^{1,0}(y) \rangle \ R(x,y)))$.

In the encoding module, APE v2 combines the domain model and workflow specification and translates the resulting synthesis problem into a CNF (propositional logic) format. The translation is presented in detail in Chapter 4.

The synthesis module takes the resulting propositional formula and passes it to a SAT (MiniSAT [38]) solver. The APE v2 implementation retains the iterative deepening approach used in APE [75], as it was demonstrated to be effective in practice. This approach synthesises the workflows involving the smallest number of operations first, which usually corresponds to the user's expectations.

Finally, the post-processing module parses and translates the solutions provided by the SAT solver to actual workflows. These can be output in a plain text form, as PNG images depicting the structure of the workflow, or in the Common Workflow Language format (CWL [8]), facilitating subsequent use in various scientific workflow management systems.

5.2 Domain Model

As described in the previous chapters, the semantic domain model constitutes the knowledge base on which APE relies for the automated composition of workflows. It comprises a domain ontology and a collection of semantically annotated tools. Additionally, the domain model might include SLTL^x constraints to express further domain knowledge or rules. These are referred to as *domain constraints*.

Name	Operation	Data input (type / format)	Data output (type / format)
Comet	Peptide database search	**Mass spectrum**	**Peptide identification**
		mzML or **mzXML**	**pepXML**
msconvert	Formatting Filtering	**Mass spectrum**	**Mass spectrum**
		MGF or **mzXML** or **mzML**	**MGF** or **mzXML** or **mzML**
Peptide Prophet	Peptide identification Statistical modelling	**Peptide identification**	**Peptide identification**
		pepXML or **mzIdentML**	**pepXML**
rt4	Retention time prediction	**Peptide property**	**Amino acid index**
		TSV or **pepXML**	**TSV** or **XML**
xml2tsv	Conversion	**Peptide identification**	**Peptide identification**
		mzIdentML	**TSV**
SSRCalc	Retention time prediction	**Peptide property**	**Amino acid index**
		Textual format or **TSV**	**Textual format**
...			

Table 5.1: Fragment of an annotated set of bioinformatics tools [66].

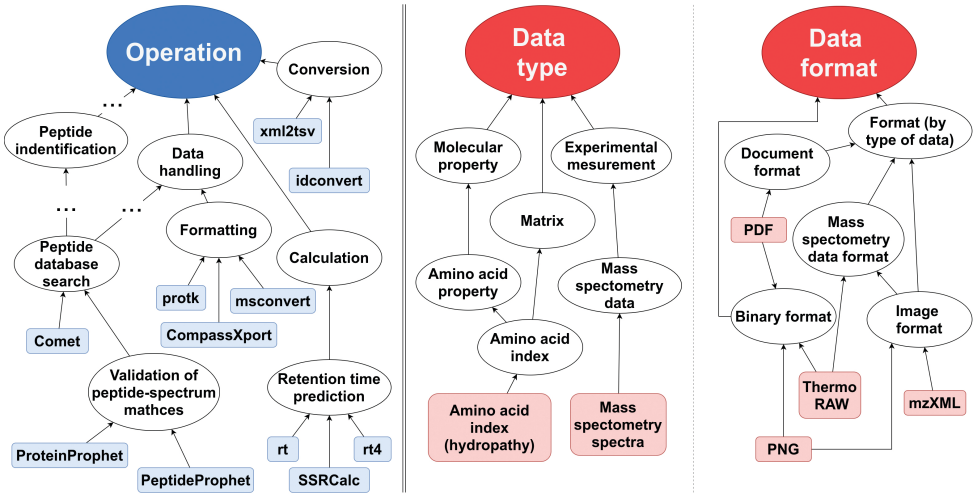


Figure 5.2: Fragment of a bioinformatics domain ontology.

For example, Figure 5.2 and Table 5.1 show fragments of a bioinformatics domain model from our recent case study on automated workflow composition in proteomics [76, 111]. The domain ontology (see Figure 5.2) was directly derived from the popular bioinformatics data and methods ontology EDAM [64]. Table 5.1 shows a few tool annotations from the same case study. Each tool is semantically annotated with the operation(s) it performs and its input and output data types and formats, using terms from the respective taxonomies. These annotations were directly derived from the bio.tools registry [65, 66], a large collection of EDAM-annotated bioinformatics tools. Note that in this example, two dimensions (type and format) are used for the annotation of the input and output data. Other applications need only one (e.g., format [73]), and yet others have more than two required dimensions [84]. Hence, APE supports the use of multiple disjoint taxonomy trees to represent the required dimensions of data characterization.

Technically, we rely on existing and (de facto) standard formalisms for the representation of the domain model. APE loads the domain ontology from a file in Web Ontology Language (OWL) format[1]. The tool annotations are represented in JavaScript Object Notation (JSON) format, following the schema that is used in the bio.tools registry [18].

5.3 Automated Workflow Composition

Once the domain model has been configured, APE is ready to be used for automated workflow composition. Therefore, the user specifies the workflow inputs, intended outputs and additional constraints that the workflow has to fulfil. For example (as

[1]The APE framework supports RDF/XML, OWL/XML, OWL Functional Syntax, Manchester OWL Syntax, Turtle, KRSS and OBO Flat ontology file formats. The parsing is handled by OWL API v5.1.20 (`https://github.com/owlcs/owlapi/wiki`).

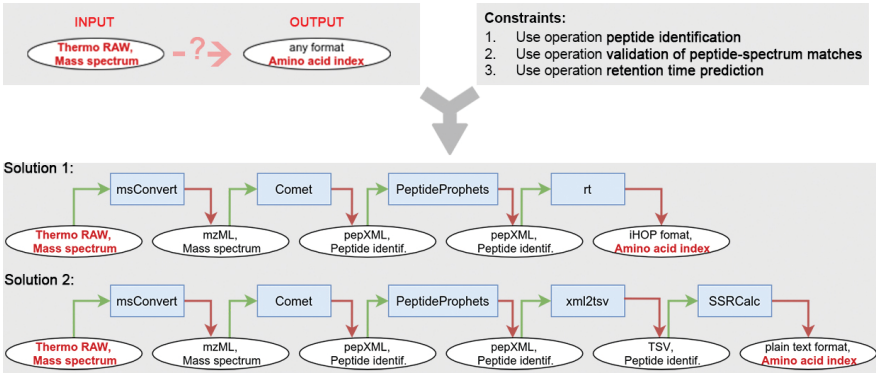


Figure 5.3: Automated composition of a proteomics workflow.

illustrated in Figure 5.3), one workflow specification from the proteomics case study consists of "Mass spectrum" type in "Thermo RAW format" as input, "Amino acid index" (in any format) as output, and constraints specifying to use tools that perform the operations "peptide identification", "validation of peptide spectrum matches" and "retention time prediction" (constraint template "Use operation *X*"). These operations are abstract terms from the ontology, known to scientists from the domain. This shows that formulating such constraints does not require knowledge of all available tools that fit the description. Based on the given specification, APE synthesises workflows that fulfil the specification by construction. Figure 5.3 shows two of many possible workflow solutions for the example specification.

Automated workflow composition with APE can be performed through its command line interface (CLI), its application programming interface (API) or the latest graphical browser-based interface (GUI). While the CLI provides a simple means to interact and experiment with the system, the API provides more flexibility and control over the synthesis process. It can also be used to integrate APE's functionality into other systems. The GUI on the other hand provides a more intuitive interface, suitable for new users. On top of it, it features improvements in the visualisation of the solutions.

5.3.1 Command Line Interface (CLI)

When running `APE-2.0.0.jar` from the command line, the program requires a configuration file as a parameter to execute the complete automated workflow composition process accordingly. This JSON-based configuration file provides references to all therefor required information:

1. the domain model (as described in Section 5.2), provided as a pair of well-formatted OWL and JSON files,
2. the workflow specification provided as a list of workflow inputs/outputs and template-based workflow constraints, and
3. parameters for the synthesis execution, such as the number of desired solutions, output directory, system configurations, etc.

Synthesised solutions are provided by APE v2 in the output directory. Each solu-

tion can be provided as a text file that describes the steps of the workflow, as a PNG figure that illustrates the workflow structure, and in the CWL (Common Workflow Language) [8] format.

5.3.2 Application Programming Interface (API)

Like the CLI, the APE API relies on a configuration file that references the domain ontology, tool annotations, workflow specification and execution parameters. However, the API allows the user to edit this file programmatically, e.g., to add constraints or to change execution parameters dynamically. This is useful, for instance, for providing more interactive user interfaces or for systematically exploring and evaluating workflow synthesis results for varying specifications and execution parameters.

```
JSONObject apeConfig = Utils.generateGeneralConfiguration();
apeConfig.put("ontology_path", "./EDAM.owl");
apeConfig.put("tool_annotations_path", "./biotools.json");
APE apeFramework = new APE(apeConfig);
APERunConfig runConfig = Utils.parseJson("./runConfig.json");
runCongif.setMaxNoSolutions(10);
SolutionsList solutions =
apeFramework.runSynthesis(runConfig);
apeFramework.writeSolutionToFile(solutions);
apeFramework.writeDataFlowGraphs(solutions);
```

Listing 5.1: APE v2 API calls used to synthesise workflows and save solution.

Listing 5.1 shows a small example of using the APE API for synthesising a set of workflows similar to the example in Figure 5.3. First, the paths to the domain ontology and tool annotation files are added to the APE configuration object. Then a new instance of the APE framework is created based on the configuration, and the workflow synthesis algorithm is executed with the provided run configuration. The result of the synthesis run is a list of solutions obtained from the SAT solver, which are written into the output directory in textual and graphical (data-flow) format.

The APE API provides further functionality, allowing for more fine-grained interaction with the APE framework. Figure 5.4 outlines the API, focusing for brevity on the most relevant fields and functions. The *ConstraintFactory* and *Constraint* classes allow for the retrieval of constraint templates and the addition of new or removal of existing constraints, thus further constraining or loosening the specification, respectively. As shown in the example code (Listing 5.1), the *APE* class constitutes the main interface for interaction with the framework. It is used to define the execution parameters as well as the output formats. Once the library has generated the solutions, they are provided as a list of *SolutionWorkflows*. Each solution is represented as a directed graph that comprises type and tool nodes (internally named modules). The interface for working with the workflow solutions (further elaborated in the next section) is provided by the classes *SolutionWorkflow*, *TypeNode* (representing type instances) and *ModuleNode* (representing tool instances).

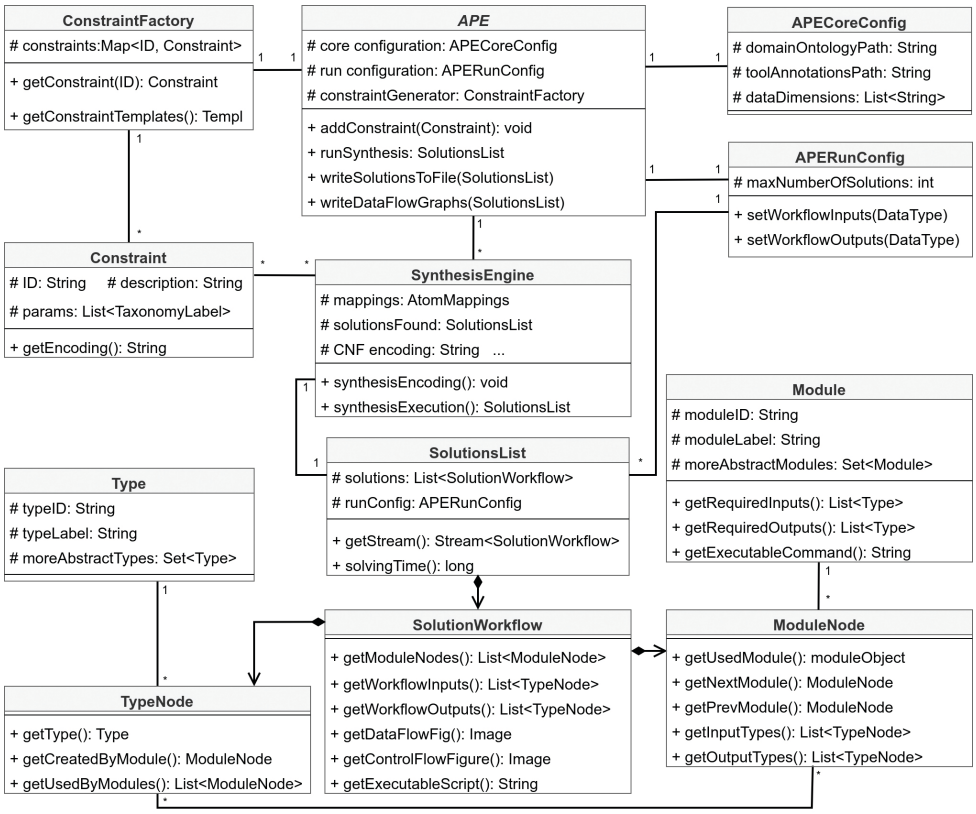


Figure 5.4: Fragment of the APE v2 API.

5.3.3 Graphical Browser-based Interface (GUI)

The APE CLI and API aim to be easy-to-use, but clearly target a tech-savvy audience with a certain level of coding and/or scripting confidence. To reach a broader audience, an intuitive interface that can be used without technical experience or specific training is required. I recently co-initiated a (Software) project[2] and co-supervised a group of 10 students to develop APE Web [55], a graphical interface built on top of APE API. APE Web is available as an open source project[3], while a running instance is hosted at `https://ape.science.uu.nl/`.

The platform provides the automated workflow composition functionality of APE through a browser-based interface. The composition can be performed over any of the available domains and allows as much freedom in parametrisation as when using the CLI version while utilising benefits of the graphical interface (see Figure 5.5a). APE Web goes even a step further, as it provides a "constraint sketcher" interface,

[2]As part of the Software Project at Utrecht University, students develop software in a professional setting. They carry out an assignment for a client in teams. The project serves as a graduation assignment for the Bachelor's degree in Computer Science.

[3]`https://github.com/sanctuuary/APE-Web`

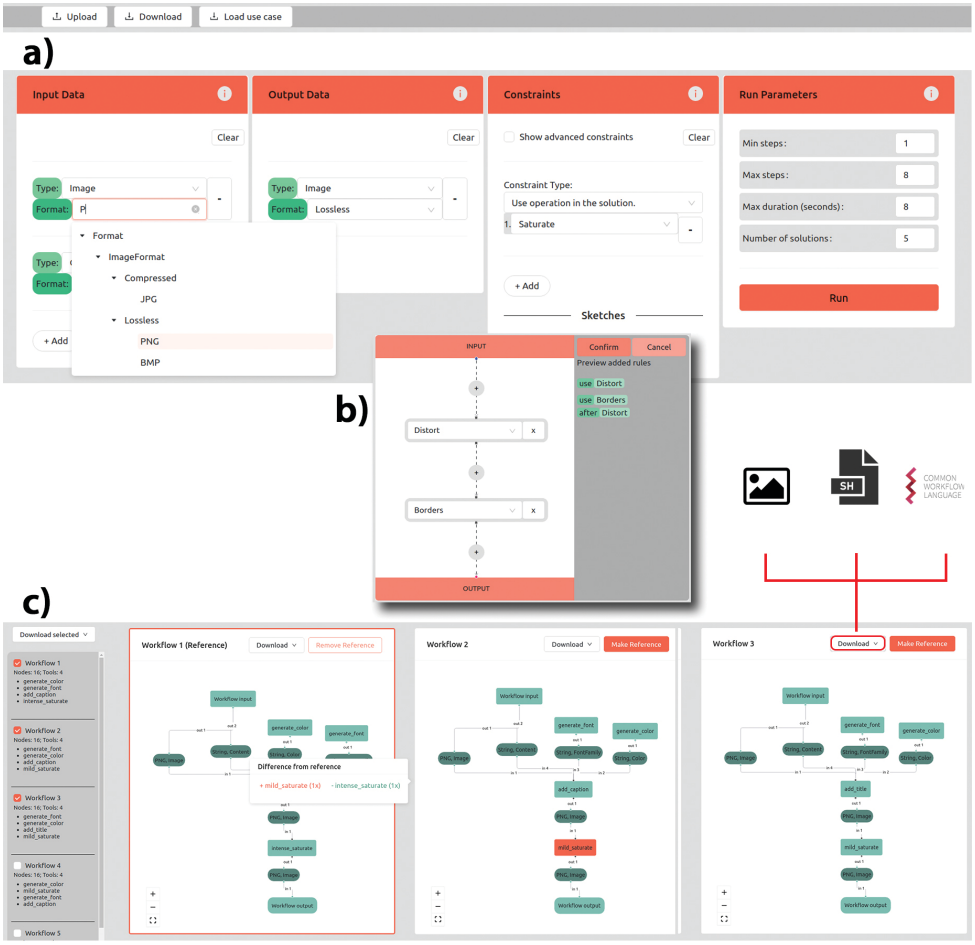


Figure 5.5: Workflow exploration interface within APE Web

which allows users to graphically sketch a workflow structure which is then translated into a set of constraints (see Figure 5.5b). Once the user performs the run, the graphical interface is used to improve the visualisation of the solutions by highlighting differences between the visualised candidate solutions (see Figure 5.5c).

Finally, the platform provides a user-based system for sharing semantic domain annotations. Therefore, users can both, create new domains, as well as extend and use existing domains to automatically compose new workflows.

5.4 Workflow Implementation

As mentioned above, APE provides functionality for exporting the synthesised workflows as textual representations, in the form of (data-flow and control-flow) graphs

and as executable shell scripts. In practice, it is often desirable to implement workflows in one of the languages used by popular workflow management systems (e.g., KNIME [16], Galaxy [2], Apache Taverna [147], Kepler [62]), to be able to execute them with the respective workflow engines. Given a large number of existing workflow languages, it is however not feasible for APE to provide ready-to-use export functionality for all of them. Instead, the information contained in APE's own workflow representation is used to create workflows in other languages. In the following sections, we describe the APE workflow format and demonstrate how it is used to create corresponding workflows in CWL. The mapping process described in this paper can furthermore serve as a template for the translation of APE results to other workflow formats, such as NextFlow [33], SnakeMake [82] or the Workflow Description Language (WDL) [148].

5.4.1 APE workflow format

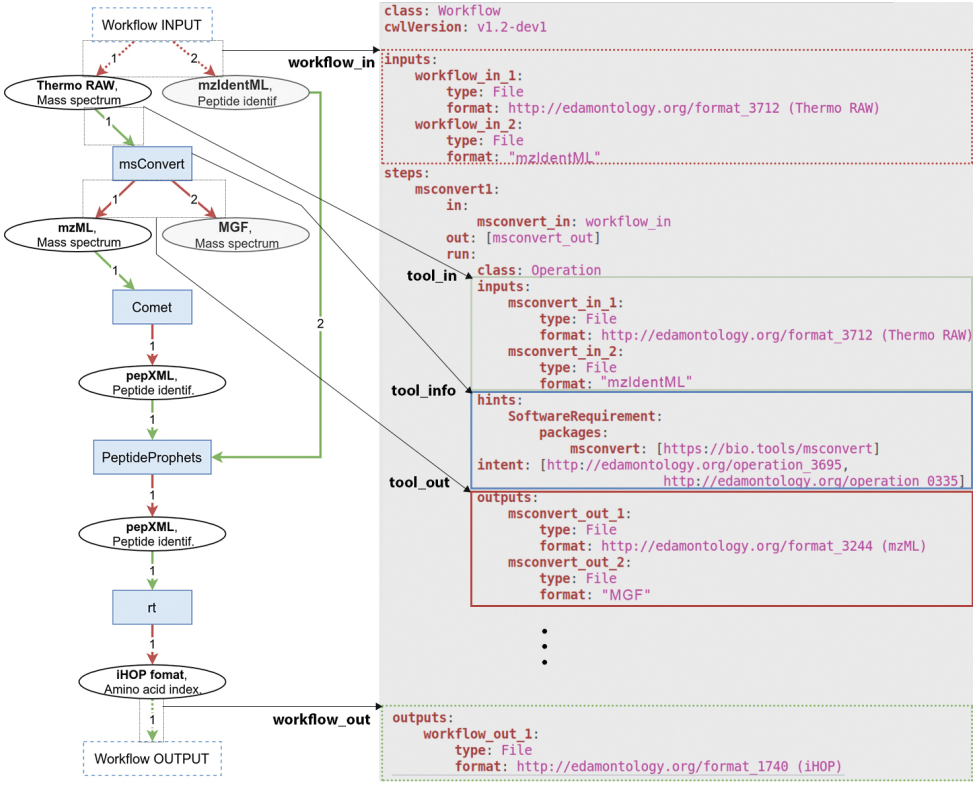


Figure 5.6: Workflow in APE's native format (left) and the corresponding workflow in CWL (right).

APE represents the workflow solutions in the form of directed graphs. The left-hand side of Figure 5.6 shows an example. Nodes in the graph represent instances of data (depicted as ellipses) and executions of operations (rectangles), while the

edges represent inputs and outputs of these tools, shown as green and red arrows, respectively. Labels on the edges represent the order in which they are given as arguments to the tools. This graph provides the trace information that is needed to create the workflow in another language.

The APE API provides a set of functions to aid the interaction with the graph structure (see class *WorkflowSolution* in Figure 5.4). The workflow inputs can simply be retrieved using the corresponding function of the *SolutionWorkflow* class, which returns it as a list of *TypeNodes*. Generally, each *TypeNode* comprises concrete data *Type* that identifies it, a (possibly empty) tool node that generated it as an output, and a (possibly empty) list of tools that used it as an input. Furthermore, the *SolutionWorkflow* class provides a function for retrieving the tools used in the workflow as a list of *ModuleNodes* (sorted according to their order of execution), making it easy to iterate over all tools used in the workflow. Each *ModuleNode* provides information about the next and the previous *ModuleNode* in the sequence, the *TypeNodes* used as inputs and generated as outputs by the tool, as well as information about the actual tool (executable script, see class *Module*) that provides the information needed for its execution. Finally, the workflow outputs are provided in the same format as the initial inputs. Note that for this example the first proposed solution from Figure 5.3 was artificially extended with additional inputs and outputs (depicted as grey ellipses) for illustrative purposes.

5.4.2 Translation to CWL

CWL[4] [8] has recently emerged as an open standard for describing scientific workflows across platforms. It is increasingly adopted by the scientific community, with CWL support being added to popular scientific workflow management systems like, for example, Galaxy [47] and Toil [140]. CWL is a declarative language that focuses on workflows composed of command line tools. Basically, it describes a set of steps and dependencies between those steps. CWL has its roots in "make"[5] and similar tools, and like them, it determines the order of execution based on these dependencies between tasks, i.e. if there is a required order of the operations or if they can even be executed concurrently. Conveniently, the main CWL structure is quite similar to the APE workflow structure. A basic workflow (see right-hand side of Figure 5.6) comprises a configuration header, a list of workflow inputs, steps to be performed and workflow outputs. The input/output dependencies have to be explicitly defined, again in line with our data-trace workflow representation. The tools in CWL usually include a command field, explicitly defining the corresponding command line operation. In addition, they can be configured to run tools from Docker containers automatically, allowing for more flexible and scalable workflow implementations.

However, as the fully automatic configuration for execution is not always feasible, the CWL version 1.2 introduces *abstract workflows*. These workflows use descriptive containers instead of directly executable operations, and require additional (manual) configuration to become executable. The abstract containers are represented

[4] `https://www.commonwl.org/`
[5] `https://www.gnu.org/software/make/`

using the *intent* label (see Figure 5.6). Given that the functional description of tools is sufficient for workflow discovery with APE, the abstract CWL workflows match well with APE's own workflow representation. Furthermore, the bio.tools registry used as a source for the tool annotations in the aforementioned bioinformatics case study is a typical example of such a set of tools. The repository contains the semantic annotations of the tools but might still require some manual steps from the user to execute the tool itself. Hence APE discovers workflows composed of tools that are not necessarily available on the local system, potentially requiring the installation and configuration of the tools on the execution system first.

We use the following steps to translate and APE workflow into CWL format. (1) Describe the original inputs, (2) iterate through the tools in the workflow sequence and specify the inputs used and outputs generated, and finally (3) specify the workflow output list. The right-hand side of Figure 5.6 shows the CWL representation of the APE workflow on the left. To create it, first, the list of input objects is translated into a list of inputs that are annotated using their formats (see label **workflow_in** in Figure 5.6). This means that some information about the data gets lost in the translation (specifically the type description). However, at runtime the format is sufficient to perform the execution, and thus, this is not a problem. Second, each tool in the sequence is described. The description involves a definition of the inputs, outputs and tool execution specification (mappings are annotated using labels **tool_in**, **tool_out** and **tool_info**, respectively). The most important part of the step is to keep track of the exact source of the tool inputs as well as to provide a sufficient tool description that would allow for its execution. The input information is already part of the formalism, as APE v2 keeps track of data flow traces for each data instance. The only requirement is to properly use the identifiers provided when creating the mappings to CWL. Regarding the tool descriptions, as long as the provided tool annotation file contains sufficient information, it can be translated into CWL. Third, the final tool outputs need to be specified based on the given tool description (see label **workflow_out** in Figure 5.6).

5.5 Related Work

This section presents an overview of software that tackle the synthesis of scientific workflows, often specialised for a specific scientific domain. We present workflow synthesis software used in geo- and life-sciences, as the domains correspond to case studies that utilise APE, elaborated in Chapter 6.

The Workflow INstance Generation and Selection (WINGS) workflow system provides a scientific workflow planning approach which is not domain specific. The approach takes an abstract/template workflow as the basis and uses it to automatically instantiate a concrete and executable workflow. This is possible as the approach provides a high-level semantic workflow representation, which can be automatically configured into executable workflows. In addition, it allows the usage of abstract steps in the workflow, which are implemented by suitable tools or sub-workflows. The implementations can be manually set and exchanged, allowing for quick generation and comparison of workflow variants.

5.5.1 Geo-service Composition and Geo-ontologies

Automated program and workflow composition is a challenging and active field of research in computer science [53], but it has not been intensively studied in the geospatial domain so far. Though tool ontologies [4] and abstract GIS operations [27] have been known for decades, they do not seem to have matured to the stage of being used for automated workflow composition. Still, we can distinguish a few different approaches aiming to simplify the creation of GIS workflows. Most of them agree that an information ontology is a suitable formalism for structuring existing data types and operations [10, 57, 94, 139, 149, 150]. This is justified by the fact that different tasks may require different levels of constraints and explanations [136], both being provided by an ontology. The existing approaches can be classified according to their preferential focus on the workflow synthesis process.

Some approaches provide an intuitive interface for helping users discover GIS tools and data sources for workflow composition [10, 68, 94, 143]. [107] recently proposed hierarchical profiles for the service discovery. These approaches still rely on a manual workflow composition, similar to workflow management approaches [96, 139]. Their focus is on using formal semantics to simplify the transition process between data sources. The same holds for Linked Data based workflow repositories [117]. Some approaches propose *task-centred ontologies* for the service chaining and data retrieval [143], and use it to retrieve and invoke workflows from a knowledge base [133, 151].

Other works[10, 57, 150] aim at automating the process of *GIS workflow composition* itself. Most of these authors focus on the semantic discovery of individual operations from a knowledge base, based on either formal input and output specifications [10, 40, 98], or based on tool thesauri [21, 57, 58]. Although operation discovery is a crucial step in workflow discovery, there is still a need for combining the discovered operations in executable workflows. Yue et al. [150] address the type chaining problem and provide automated discovery of chains of operations, based on their input/output specifications. Farnaghi and Mansourian [39] use a planning algorithm for automatically finding solutions to the sheltering problem in disaster management. These latter approaches are comparable to the technical problem we address. Yet, from an ontological viewpoint, they seem to lack a crucial distinction between semantic (conceptual) and syntactic (format) data properties [85, 86]. This distinction is seldom drawn, yet it is required to capture how concepts can be represented by different geodata formats.

5.5.2 Workflow Synthesis in Life Sciences

Life sciences show a high quality of semantic annotations when compared to other scientific domains, as contemporary life scientists rely substantially on computational tools and processes [92]. For example, as of July 1, 2022 the EDAM ontology [67] classifies 3,483 bioinformatics terms[6] and the bio.tools registry of software in the life sciences [66] comprises 25,411 life science software annotations. In addition, the domain introduces the SADI (Semantic Automated Discovery and Integration) framework [144] as a standard for creating Semantic Web Services and a

[6] `https://bioportal.bioontology.org/ontologies/EDAM`

design pattern for the formal description of the service interfaces. The SADI registry, a collection of the mentioned service descriptions, comprises thousands of services (Data as a Service (DaaS)/Application as a Service (AaaS)).

The availability of these semantic annotations reflects the maturity of approaches that provide automation in workflow creation in the field. An early example is the ontology-driven assisted web service composition facilitated by BioMoby [34], which was integrated into the Taverna workflow system to guide the construction of workflows [145]. The goal of the approach is to simplify the interactive service composition of BioMoby services. At each step of the workflow construction process, only services which are compatible and likely to be useful are displayed. Similarly, the tool recommender system in Galaxy [87] focuses on finding suitable tools to aid the user while developing a workflow. Galaxy is a popular web-based platform for high-throughput sequencing data and other big data analyses in bioinformatics. The platform comprises more than 2,000 tools that can be hard to explore. The recommender system suggests possible next tools in an incomplete workflow, to simplify the workflow construction. The approach is based on a deep learning algorithm that utilises existing workflows in the Galaxy platform.

Magallanes [116] is a Java library used to help researchers to discover bioinformatics web services and associated data types. An important feature of Magallanes is its ability to chain available and compatible web services into workflows, according to the desired output. Magallanes provides an application programming (API) and a graphic user interface (GUI).

Finally, SHARE [138] and HYDRA [13, 115] are specialised query engines used to synthesise and execute workflows over the SADI registries. The engines use SPARQL queries to reason over the registry. SADI query engine checks the input/output descriptions to ensure compatibility between services. The process is used to discover SADI web services that match the required data transformations. In addition to writing SPARQL queries directly, SHARE supports SPARQL Assist, a language-neutral query composer, while HYDRA provides a keyword-based and graphical interface. SHARE and HYDRA provide fully automated scientific workflow composition and execution. A given query is used to compose required services into a workflow, which is executed subsequently. Unlike the other approaches in the category, this approach does not provide the workflows to the user. Instead, the result of the synthesis is the computed output data itself. This level of automation is possible due to well-annotated and curated semantic annotations of both, operations (AaaS) and data sources (DaaS) provided by the SADI registry.

The presented approaches are often hard to compare as they focus on a different part of the workflow life-cycle [92], as well as different domains. Some systems [34, 87] provide assistance while choosing concrete workflow steps, other [46, 74] aid in setting up the “abstract” workflows, before they can be implemented, while a few [13, 138] automate the whole process omitting the workflow altogether. Similarly, the theoretical approaches (e.g., [15, 23, 61]), presented in Chapter 3, aim to solve complex and universal synthesis problems, assuming that the required implementations and/or domain annotations are available. While, the more practical solutions (e.g., [39, 116, 150]) often tailor the implementation for a specific domain and available resources. Therefore, the comparison of such formalisms is challeng-

ing, if not impossible.

Interestingly, to the best of our knowledge, the existing approaches do not cover automated workflow benchmarking yet, which is however essential for bringing automatically created workflows to the production stage.

CHAPTER 6

Case Studies

Abstract - To assess the benefits of the proposed workflow synthesis approach, we evaluate it in the current setting of computational semantic domain annotations. We focus on life- and geo-science domains as they show notable progress in semantic annotations of the respective domain terminology, data types and operations (e.g., EDAM and bio.tools, and CCD ontology, respectively). This chapter presents case studies we conducted in collaboration with scientists from life- and geo-sciences.

The chapter presents four different case studies. While they all demonstrate use cases for the presented synthesis approach, each of them aims to assess a specific set of features of the APE v2 framework, and the underlying SLTL^x-based synthesis approach. (1) A geovisualisation case study, presents an illustrative use case that demonstrates distinguishing data objects as a key feature of the workflow synthesis approach. The study demonstrates a domain set-up process, as it provides a new semantic domain model for a set of geovisualisation tools, due to the lack of existing formal annotations. (2) A Geo-Analytical Concept Question Answering case study, focuses on a domain model that comprises multiple semantic data dimensions, as well as utilises the data-tool dependency constraints to reduce the redundancies within the synthesised workflows. We present a domain model and problem specifications in collaboration with domain experts, and evaluate the quality of the solutions provided by the APE framework. (3) The proteomics case study presents the performance and synthesis quality of our synthesis framework as an "off-the-shelf" tool when applied to existing large domain models in the proteomics domain. This is possible due to the availability of the community-curated semantic domain annotations, provided by the EDAM ontology and the bio.tools registry. (4) The Geo-Event Question Answering preliminary case study demonstrates the potential usage of APE within a larger geo-event question answering framework. In addition, it proposes two post-processing techniques for improving the generated results.

The chapter concludes with a discussion of the assessed synthesis features and their benefits.

This chapter is partially based on the following publications:

Kasalica, V., Schwämmle, V., Palmblad, M., Ison, J. & Lamprecht, A.-L., "APE in the Wild: Automated Exploration of Proteomics Workflows in the bio.tools Registry", *Journal of Proteome Research*, vol. 20, no. 4, 2021, PMID: 33720735, pp. 2157–2165, DOI: 10.1021/acs.jproteome.0c00983, eprint: https://doi.org/10.1021/acs.jproteome.0c00983, URL: https://doi.org/10.1021/acs.jproteome.0c00983.

Kasalica, V. & Lamprecht, A.-L., "Workflow Discovery Through Semantic Constraints: A Geovisualization Case Study", in: *Computational Science and Its Applications – ICCSA 2019*, Cham: Springer International Publishing, 2019, pp. 473–488, ISBN: 978-3-030-24302-9.

Scheider, S., Meerlo, R., Kasalica, V. & Lamprecht, A.-L., "Ontology of Core Concept Data Types for Answering Geo-Analytical Questions", *Journal of Spatial Information Science*, vol. 2020, no. 20, 2020, pp. 167–201, DOI: 10.5311/JOSIS.2020.20.555, URL: https://digitalcommons.library.umaine.edu/josis/vol2020/iss20/2 (visited on 02/04/2022).

Kruiger, J. F., Kasalica, V., Meerlo, R., Lamprecht, A.-L., Nyamsuren, E. & Scheider, S., "Loose programming of GIS workflows with geo-analytical concepts", *Transactions in GIS*, vol. 25, no. 1, 2021, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.12692, pp. 424–449, DOI: 10.1111/tgis.12692, URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/tgis.12692 (visited on 09/02/2021).

Kazemi Beydokhti, M., Duckham, M., Griffin, A. & Kasalica, V., "Geo-Event Question Answering Systems: A Preliminary Research Study", Sept. 2021, DOI: 10.25436/E2KW2T, URL: https://escholarship.org/uc/item/9cs309kd (visited on 11/01/2021).

This chapter aims to demonstrate the benefits of using APE v2 [74] and its SLTL^x-based workflow synthesis [72] in practice. We present case studies from geo- and life-sciences that use APE v2 to automatically compose workflows and answer domain-specific questions. Each of the case studies aims to evaluate a specific feature of the APE v2 framework (see Table 6.1). The case studies are summarised as follows.

Geovisualisation The initial illustrative case study [73] aims to demonstrate the ability of the SLTL^x-based formalism to *distinguish data objects*, and define *object specific constraints*. The study introduces a new purpose-made domain model. It demonstrates *setting-up domain* taxonomies, tool annotations (including executable *shell* commands) and specifying domain-specific constraints. The expected workflow comprises numerous data files of the same data type, and as such provides an ideal setting to assess the benefits of the data object distinction. Furthermore, the case study illustrates the *iterative workflow synthesis* approach, where the user tries to narrow down the desired specification of the problem in a few iterative steps. Instead of providing all the constraints at once, the user initially provides a simple specification, that gets extended after each synthesis run, until the desired results have been reached. This allows us to automatically compose fairly complex solutions. See Section 6.1 for more information.

Geo-Analytical Concepts The case studies we did with collaborators from geosciences [84, 120] aim to assess the benefits of *multiple semantic data dimensions*. They focus on the automated composition of workflows over a new Core Concept Data types (CCD) ontology of geo-analytical concepts [120]. Due to a large number of similar semantic tool annotations and potential redundancies in the suggested solutions, the case study provides an optimal setting for assessing the benefits of the *data-tool dependency constraints*. Finally, the study demonstrates *benefits of a well-annotated domain vocabulary*, compared to a benchmark classification generated from common data types. See Section 6.2 for more information.

Life Sciences - Proteomics Data Analysis The case study in the proteomics[1] domain [76] assesses the quality the *workflow synthesis approach as an "off-the-shelf" synthesiser*, i.e., in an existing semantically annotated and curated domain (EDAM [67] ontology and bio.tools [66] registry of tools). It displays the quality of solutions generated by the synthesis approach in the domain, without any modification of / improvements to the semantic domain annotations. The results of the case study indicate that the importance of the automated workflow composition will grow further as scientific domains adopt or improve their semantic annotations. See Section 6.3 for more information.

Geo-Events The preliminary study concerning geo-events [79] aims to show the potential of the workflow synthesis using the *APE within a question answering framework*. The study goes a step further and proposes two post-processing steps. Their aim is to improve the quality of the question answers, by grouping together the candidate solutions according to the corresponding specific criteria. See Section 6.4 for more information.

[1] Proteomics is a branch of life sciences that focuses on protein research.

Case study	Collaborators	Semantic tool annotations	Semantic taxonomy annotations	Assessed APE features
Geovisualisation	-	*user defined*	*user defined*	• distinguishing data objects • domain set-up process • iterative workflow synthesis
Geo-Analytical QA	Scheider, S.,[a] Kruiger, J.,[a] Meerlo, R.[a]	*user defined*	**CCD Ontology**	• data-tool dependencies • modelling of multiple dimensions • synthesis within a well annotated domain model
Proteomics Data Analysis	Schwämmle, V.,[b] Palmblad, M.,[c] Ison, J.[d]	**bio.tools registry**	**EDAM Ontology**	• performance as an "off-the-shelf" synthesiser • synthesis quality over large real life domains
Geo-Event QA	Kazemi Beydokhtie, M.,[e] Duckham, M.,[e] Grifn, A.[e]	*user defined*	*Extended* **CCD Ontology**	• APE within a larger framework

Table 6.1: Overview of the Case studies.

[a]Department of Human Geography and Spatial Planning, Utrecht University, Netherlands
[b]Department of Biochemistry and Molecular Biology, University of Southern Denmark, Denmark
[c]Center for Proteomics and Metabolomics, Leiden University Medical Center, Leiden, Netherlands
[d]Institut Français de Bioinformatique, CNRS, Crémieux, France
[e]Department of Geospatial Science, RMIT University, Melbourne, Australia

The chapter concludes with a discussion of the synthesis results evaluated in the presented case studies.

6.1 Geovisualisation

This section presents the case study from the field of geovisualisation, that was used as an illustrative example in Chapters 1 and 3. The study focuses on the automated creation of a topographic map of the Netherlands that depicts waterbird movements [73]. It presents the benefits of the SLTL^x-based workflow synthesis in a domain that comprises suitable technical vocabularies and tool annotations.

The specification refinement in this section goes a step further than our previous work [73]. Since the initial case study, the APE framework has significantly changed. Most notably, the underlying logic (SLTL^x) now supports the data object-specific constraints. This allows us to generate the workflows without splitting them into smaller fragments[2]. This eliminates the manual post-processing steps included in the original study.

The case study is structured as follows. Section 6.1.1 describes the application example, i.e., the creation of a map that presents animal tracking data and topographical features of the area, which we address in this study. Section 6.1.2 describes the domain model that we set up for the use case. Section 6.1.3 demonstrates the incremental synthesis process. Finally, Section 6.1.4 summarises the case study.

All data used to run the case study and generate the results is available at `https://github.com/sanctuuary/APE_UseCases/tree/master/GeoGMT`.

6.1.1 Problem Description

Cartographic workflows are implemented by sequences of tools that perform the individual elementary operations of the map creation process. There are many different Geographic Information System (GIS) tools available that can be used for map creation, such as the tools provided within ArcGIS [70], the Geospatial Data Abstraction Library (GDAL) [45], the CSISS Geospatial Web Services[3] or the already mentioned Generic Mapping Tools (GMT). For the purpose of this case study, we focus on the GMT, but the same approach can in principle be applied to any other of the aforementioned tools. The main reasons for choosing the GMT for this case study are threefold. (1) It is a fairly generic tool-set, as it supports various data input formats. (2) The concise and well-structured documentation of the tools, provides a rich source of information as a basis for semantic domain modelling. (3) The modularity of the GMT makes it highly suitable for workflow composition, as the individually accessible elementary operations allow firm control over the entire plotting process.

Our workflow use case is about discovering a computational pipeline that creates a topographic map depicting waterbird movement patterns in the Netherlands (Fig-

[2] In the original study, due to the expressive power of the SLTL formalism, each incremental (extension) step was synthesised as a separate workflow, and the workflows were post-processed and manually merged together.

[3] `http://geobrain.csiss.gmu.edu/grassweb/manuals/index.html`

ure 6.1). Wildlife tracking is an important process for biologists and environmental scientists to improve their understanding of animal behaviour. Movement behaviour gives insight into the ecology of animals, their interaction with other organisms, and their effect on the ecosystem dynamics [36]. It can help to predict how environmental changes can affect their role in the ecosystem. The use case combines tools from the GMT collection with data from the Movebank [83] online database of animal tracking data. It is used to help animal tracking researchers to manage, share, protect, analyse, and archive their data. The database supports multiple sources of data, including the integration with the Argos system (`http://www.argos-system.org`), which is a leading source of wildlife tracking data worldwide [146]. This type of source provides diverse, robust and high-quality data, which is however often difficult to exploit and plot on a map, especially for ornithologists and other field researchers who are not very familiar with GIS tools [29, 37]. The data we focus on in our example was used to find correspondences between movement patterns of a keystone waterbird species and the landscape configuration [80]. The data represents mallard movement patterns in the Netherlands [80].

6.1.2 Semantic Domain Modelling

In this section, we describe the domain model that we set up for the use case scenario. It comprises a tool and a type taxonomy for defining a controlled vocabulary for the operations and data types in the domain. In addition, it incorporates tool definitions using the terms provided by the taxonomies and a set of so-called domain constraints that express additional relevant knowledge. Overall, the GMT comprises over 100 different tools that can be parameterised on an elementary level, resulting in even more possible operations that would have to be defined as tools. For simplicity and conciseness, we have hence limited the domain model to the GMT tools that are relevant to the example scenario.

Figures 6.2 and 6.3 show the type and the tool taxonomy, respectively. The classification of tools and types in these taxonomies is essential for the effective usage of the synthesis algorithm, as mentionedin Chapter 5. To the best of our knowledge, there is no similarly structured classification model available for GMT, so we defined new taxonomies for this purpose. The data types used by the GMT tools are well documented [142] and at the same time not overly complex. Therefore, the type taxonomy follows straightforwardly from the documentation and covers all of the data types mentioned in the documentation. The documentation covers one dimension of data, which is sufficient to model the domain. Additional dimensions might be needed in case the domain is to be extended to a more general geovisualisation context. The tool taxonomy, on the other hand, focuses on classifying just a part of the GMT in a detailed manner. The idea was to classify the tools occurring in our scenario in a way that would allow a workflow developer a simplified workflow specification, using the newly introduced abstract classes.

For example, the tool *pscoast_s* in the lower right corner of the tool taxonomy (Figure 6.2) is an implementation of the GMT command used to automatically colour water surfaces on the map. The tool taxonomy allows us to abstract from the concrete tool and refers to it simply as a *Draw water mass* tool. Another abstract class groups the tools that are used to *Draw water*. As we go up the taxonomy tree we

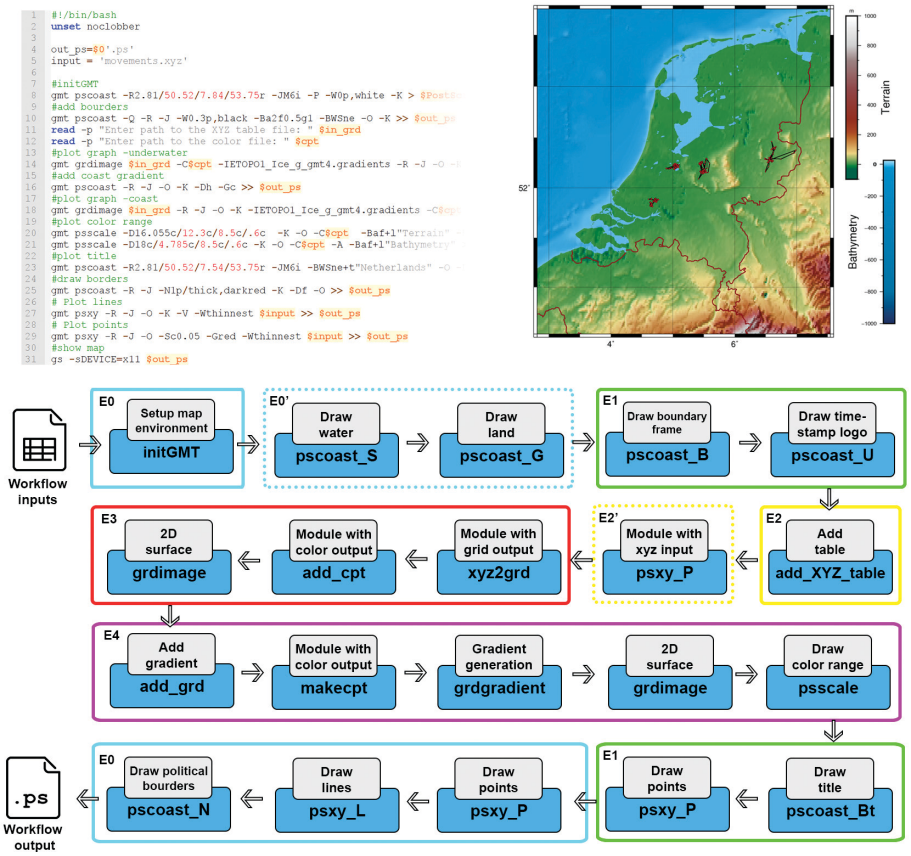


Figure 6.1: Incremental development of the workflow (extension steps E0-E4), the corresponding generated GMT script, and the map created by its execution.

can see that our initial tool belongs to the class of tools used to create *Basemaps*, eventually characterising all the tools used for *Plot creation*. Finally, every instance of the tool taxonomy is, naturally, considered to be a *Tool*. Every tool and data type in the taxonomies is classified in a similar manner.

Table 6.2 lists the tools defined for the domain model, each with its name, function description and its (possibly empty) sets of input and output types. The idea is to use the tools as elemental components of our workflows, that can be easily annotated and classified. In some cases, several tools refer to the same underlying GMT operation (*polymorphism*). For example *pscoast_B*, *pscoast_Bt*, *pscoast_U* and *pscoast_Td* all call the *pscoast* operation, but with different parameters, causing it to perform different functions.

In addition to the taxonomies and tool annotations, the domain model can comprise a set of SLTL^x formulas to express general knowledge that the synthesis algorithm also takes into account. These *domain constraints* are typically used to avoid obtaining workflows that are ambiguous, redundant, or not relevant to the domain.

Name	Description	Data type in	Data type out
add_cpt	Provide a colour palette (.cpt) file		cpt_file
add_XYZ_table	Provide an xyz table file		XYZ_table_file
add_grd	Provide a grid file		NetCDF
grdgradient	Compute directional gradient	NetCDF	Intensfile
makecpt	Make colour palette tables	cpt_file	cpt_file
xyz2grd	Convert an xyz table file to a 2-D grid file	XYZ_table_file	NetCDF
psconvert	Crop and convert PostScript files to PDF	PostScript	PDF
psconvert	Crop and convert PostScript files to PNG	PostScript	PNG
psconvert	Crop and convert PostScript files to JPEG	PostScript	JPEG
grdcontour	Contouring of 2-D gridded data sets	NetCDF	PostScript
grdview	3-D perspective imaging of 2-D gridded data	NetCDF, cpt_file	PostScript
grdview	3-D perspective imaging of 2-D gridded data	NetCDF, cpt_file, Intensfile	PostScript
grdimage	Produce colour images from 2-D gridded data	NetCDF, cpt_file	PostScript
grdimage	Produce colour images from 2-D gridded data	NetCDF, cpt_file, Intensfile	PostScript
psxy_L	Plot lines	XYZ_table_file	PostScript
psxy_P	Plot location points	XYZ_table_file	PostScript
pstext	Plot text strings on maps	XYZ_table_file	PostScript
psscale	Plot grayscale or colour scale on maps	cpt_file	PostScript
pscoast_B	Drawing the map boundaries and grid		PostScript
psbasemap_B	Drawing the map boundaries and grid		PostScript
pscoast_Bt	Write the title of the map		PostScript
psbasemap_Bt	Write the title of the map		PostScript
pscoast_U	Draw the GMT logo and the time stamp		PostScript
psbasemap_U	Draw the GMT logo and the time stamp		PostScript
pscoast_Td	Draw the windrose		PostScript
psbasemap_Td	Draw the windrose		PostScript
pscoast_G	colouring land surfaces		PostScript
pscoast_S	colouring water mass		PostScript
pscoast_I	Draw rivers		PostScript
pscoast_N	Draw political borders		PostScript
pscoast_W	Draw water borders		PostScript
pscoast_F	colouring countries for which the country codes were provided		PostScript
initGMT	Setup the GMT map environment		
gs	Tool used to display PostScript files	PostScript	

Table 6.2: Tool annotations introduced for the Geovisualisation case study.

ID	Constraints in natural-language	Constraints in SLTL^x
G1	At least one of the outputs per **tool** should be used subsequently (as tool inputs)	G (∃x (<**Tool**^{0,1}(x)> F<**Tool**^{1,0}(x)> true))
G2	If **Data processing** is used, tool **Data generation** cannot be used subsequently	G(¬<**Data processing**^{0,0}()> true | X G ¬<**Data generation**^{0,0}()> true
	...	
G14	If **Data presentation** is used, tool **Plot creation** cannot be used subsequently	G(¬<**Data presentation**^{0,0}()> true | X G ¬<**Plot creation**^{0,0}()> true
G15	Do not use tool **3D surfaces**	G ¬ <**3D surfaces**^{0,0}()> true
G16	Use the data type **Plots**	F ∃ x (**Plots**(x))
G17	Use tool **Map environment set-up**	F <**Map environment set-up**^{0,0}()> true

Table 6.3: Domain constraints for the Geovisualisation case study, where "**Tool**" is the most abstract concept in the Tool Taxonomy.

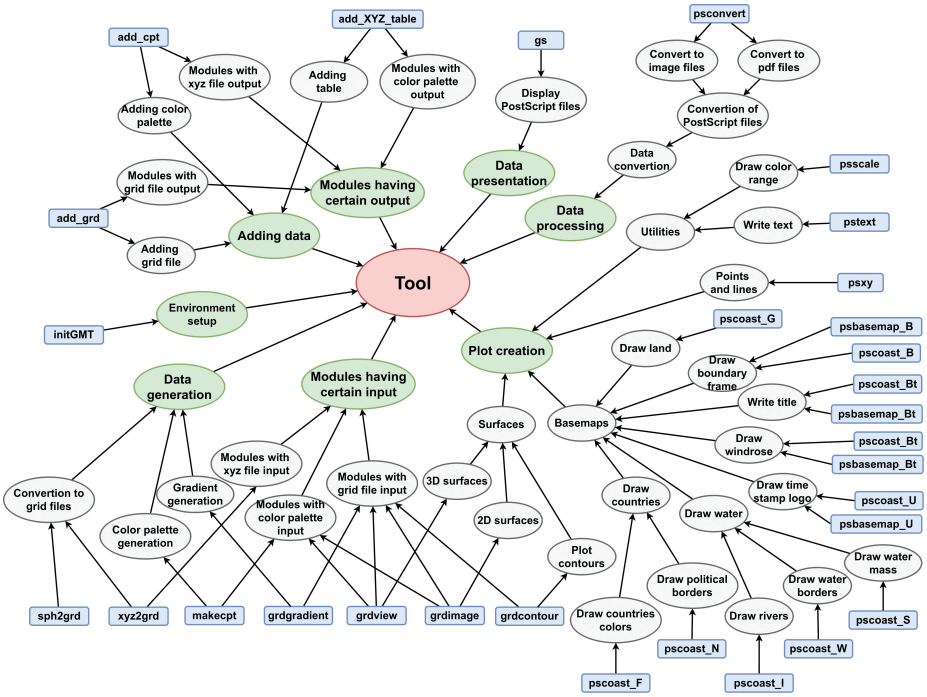


Figure 6.2: Tool Taxonomy in the Geovisualisation domain.

SLTL^x allows us to use concrete as well as abstract tools from the tool taxonomy to constrain the allowed transitions. For example, $< tool_c^{0,0}() > \phi$ requires ϕ to hold in a state reachable by the tool $tool_c$. Similarly, concrete, as well as abstract types from the type taxonomy, can be used to describe the data types used by the tools. A detailed description of the formalism is presented in Chapter 3.

For the formulation of the constraints, knowledge of the SLTL^x syntax is not required. The APE framework provides natural-language templates for the most commonly used SLTL^x constraints. Table 6.3 (**G1**-**G17**) lists the original SLTL^x and the natural-language representations of the domain constraints defined in this study. The constraints define the basic restrictions in map generation, and thus, simplify the specification for the user. For example, constraint **G1** guarantees that creating a certain file within a workflow requires a subsequent usage of that particular file. In other words, it makes sure that the synthesis does not construct a program that loads or generates files that are not being used in the process. Similarly, using constraints **G2** - **G14** we ensure a correct ordering of the tools, to avoid overriding important annotation data and to prevent unnecessary permutations in the solutions. Furthermore, we target 2D and not 3D map representations and use the constraint **G15** to exclude the tools used for 3D plotting, which are represented by an abstract class *3D_surfaces* in the tool taxonomy. Finally, we want to use GMT tools and have a map as a product of each of our programs. This requirement can be fulfilled by using an abstract class from the type taxonomy, more specifically, by enforcing the usage of a

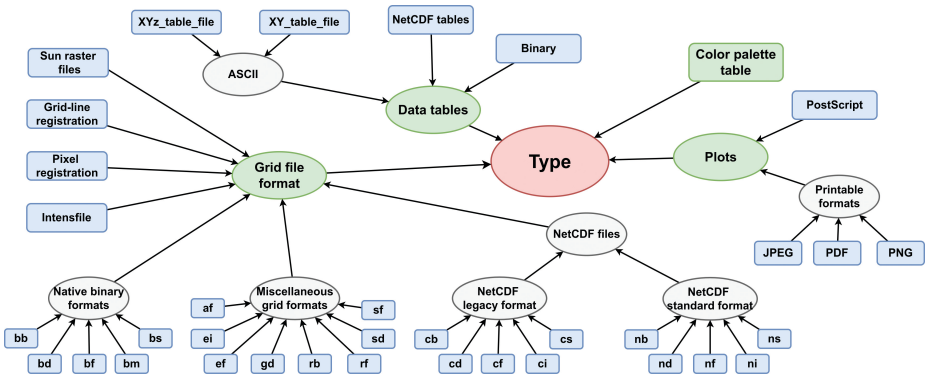


Figure 6.3: Type Taxonomy in Geovisualisation domain.

ID	Constraints
E0.1	Use tool **Draw_water**
	F <**Draw_water**^{0,0}()> true
E0.2	Use tool **Draw_land**
	F <**Draw_land**^{0,0}()> true
E0.3	Use tool **Draw_political_bourders**
	F <**Draw_political_bourders**^{0,0}()> true

Table 6.4: Initial workflow constraints.

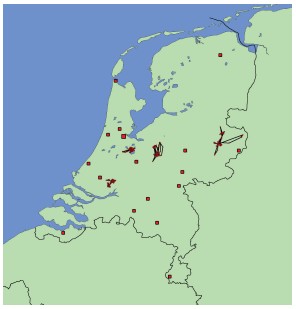


Figure 6.4: Initial workflow output.

data type *Plots* and the tool setup *initGMT*, as displayed in **G16** and **G17**. These 17 general constraints are used in all synthesis runs.

6.1.3 Synthesis Results

In this section, we focus on the workflow specification and synthesis phase and show how our framework can be applied to discover and compose a complex workflow without having to identify and connect the individual tools manually. The following sections discuss incremental synthesis steps that lead to the intended geovisualisation workflow.

Initial Workflow

Synthesis in our framework starts with two sets of data types (possibly empty), that correspond to the input and output of the workflow. In addition, SLTL^x constraints can be added to express further intents of the user. In our scenario, the initial data types are the two table formats [4] provided by the use case (coordinates of bird migration and of larger cities in the area), while the output data type corresponds to the common map representation format - PostScript. This would already provide

[4]The tables are provided in the *XYZ table* format. However, depending on the data source, table formats may differ. They can be provided in the *CSV* format, as illustrated in Chapter 3.

enough specification to get a transition from our initial data into a PostScript file. However, we would like to add some context to the plotted locations, such as distinguishing land from the water surface. Thus, we add constraints to express the first, trivial requirements that need to be fulfilled (Table 6.4). Constraint **E0.1** ensures that a tool for drawing the sea is used. *Draw_water* contains three subclasses (Figure 6.2), where each of them contains a different tool that can be used to satisfy the constraint. Moreover, constraints **E0.2** and **E0.3** follow the same logic for the drawing of the land and the political borders, respectively. These constraints are combined with the general constraints (**G1**-**G19**) to form the initial workflow specification.

With this specification as input, we start the actual workflow synthesis. Note that APE performs the search for possible workflows until the first depths where solutions are found. For the initial loose specification, our synthesis tool finds 56 solutions of length 6. That is, even though only four constraints (*G19* and *E0.1 - E0.3*) were used that enforce the use of certain tools, each solution contained at least two additional steps. The reason for this lies in the general constraint in the domain model. The existence of the input data types *XYZ_table* requires subsequent usage of a tools that would implement the XYZ table (**G1** in Table 6.3). Thus, the domain model ensures the usage of a tool that is required, but not explicitly requested by the user.

Although the number of suggested solutions is relatively large, after further investigation we notice that most of the workflows provide the same certain permutations in the operational execution of the same solution. The main points of distinction are the operations used to plot the provided two files, sometimes the coordinates would be plotted as lines and sometimes as points on the map. From the possible suggested workflows, we picked a workflow that used points to plot cities, shown in Figure 6.1 under labels **E0** and **E0'**. The workflow uses suitable tools for colouring the land and the sea, draws the political borders and uses point locations to depict the animal movement patterns and city locations. The framework provides the workflow and its implementation as a shell script, which upon execution produces the map presented in Figure 6.4.

The presented map is not yet the intended result. To add more information to the map, more steps need to be included in the workflow. One possibility is to consider synthesis solutions of lengths greater than 6. For example, we might want to use both points and lines to depict mallard movement patterns. A workflow for that case would be found at depth 7, additionally including the tool *psxy_L*. If we expand our search correspondingly, we find that there are 2300 possible workflows of this length, and their evaluation is required to find the suitable one. The evaluation of this amount of workflows is usually not feasible. An alternative approach is to reduce the size of the search space and thus the number of possible solutions found. This is accomplished by adding further constraints, describing the actually desired workflow as precisely as possible, and restarting the exploration process.

Extension 1: Annotations

As mentioned, the first map clearly lacks some information, such as annotations (frame, grid, title etc.). To properly annotate the map, we have to define corresponding tool enforcement constraints. For example, we would like to have both

ID	Constraints
E1.1	Use **Draw_lines** with *birds* data as input.
	F ∃x (**birds**(x) & <**Draw_lines**^{1,0}(x)> true)
E1.2	Use **Draw_points** with *birds* data as input.
	F ∃x (**birds**(x) & <**Draw_points**^{1,0}(x)> true)
E1.3	Use **Draw_points** with *cities* data as input.
	F ∃x (**cities**(x) & <**Draw_points**^{1,0}(x)> true)
E1.4	Use tool **Draw_boundary_frame**
	F <**Draw_boundary_frame**^{0,0}()> true
E1.5	Use tool **Write_title**
	F <**Write_title**^{0,0}()> true
E1.6	Use tool **Draw_time_stamp_logo**
	F <**Draw_time_stamp_logo**^{0,0}()> true

Table 6.5: Extension 1 constraints.

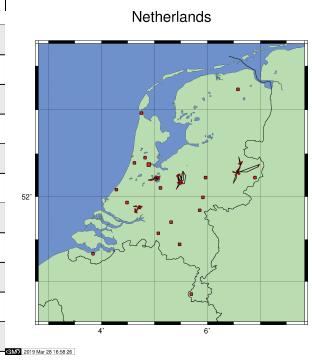


Figure 6.5: Extension 1 map.

annotations of the mallard locations as well as annotations of the path taken, to properly understand the movement patterns from the map (**E1.1** and **E1.2** in Table 6.5). In addition, we would like to make sure that the cities are plotted as points on the map (**E1.3**). We would like the map to include, the coordinates and border frame of the map (**E1.4**), the name of the area of interest (**E1.5**), the time it was created, as well as the name of the toolset used in the process (**E1.6**). Our goal is to enrich the annotation of the previously generated map, hence, extending the generated workflow with corresponding annotation tools.

Our new workflow specification consists of the constraints used before and the new set of constraints. For this specification, the synthesis returns 6,912 (shortest) solutions of length 10. Although the number of candidate solutions is too big to evaluate, they were mostly different permutations of the tools, or they incorporated different versions of the same tools (e.g. some tools have *pscoast* and *psbasemap* versions of it)[5]. Evaluation of the first 5 candidate solutions has shown that the optimal solution in this scenario corresponds to the workflow presented in Figure 6.1 under labels **E0**, **E0'** and **E1**, where the label *E1* corresponds to the newly introduced annotation tools. The execution of the workflow produces a properly annotated version of the map (Figure 6.5).

Extension 2: Providing the elevation dataset

Even though the annotated map is self-explanatory and can be presented as such, there is still room for improvement. The figure lacks information on the characteristics of the area, as the land and the sea are depicted with simple plain colours. The study [80], which motivated us to choose the corresponding waterbird tracking data in our scenario, focuses on predicting animal movement based on the landscape configuration. Therefore, it is natural to assume that the map should depict some of the topographic and bathymetric features of the Netherlands and its surroundings.

One of the ways to solve this problem is to introduce a file that contains elevation data of the region. The data would be used to plot the relief of the land and the sea.

[5]The following extension steps result in a similar number of shortest candidate solutions. Considering that they mostly introduce permutations of the same few solutions, we omit the concrete numbers.

ID	Constraints
E2.1	Use tool **Add_table**
	F <**Add_table**^{0,0}()> true
E3.1	Tool **Color_palette_generation** should generate output used by **2D_surfaces**
	F (∃ x (<**Adding_color_palette**^{0,1}(x)> F<**2D_surfaces**^{1,0}(x)> true))

Table 6.6: Extension 2 and 3 constraints.

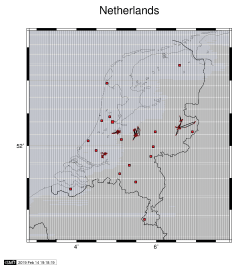


Figure 6.6: Extension 2 map.

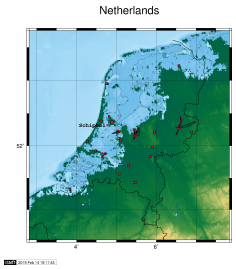


Figure 6.7: Extension 3 map.

Our initial workflow uses two plotting tools that are not required in this scenario, *pscoast_S* and *pscoast_G* (**E0'** labelled elements in Figure 6.1), as the plain colouring is be overwritten by the elevation data. Therefore, we extend the workflow obtained in the previous extension step without the two mentioned tools. The two tools can be excluded by omitting the two corresponding constraints from the specification, namely constraints *E0.1* and *E0.2*.

To accurately extend the workflow, we require the usage of the elevation data table. This is accomplished by enforcing the usage of the appropriate abstraction class from our taxonomy (**E2.1** in Table 6.6). The synthesis finds the first valid workflows at depth 10. The candidate solutions introduce the tool **add_XYZ_table** and a tool for plotting lines and points, while they exclude the operations for plotting sea and land as single colours. Based on the generated output the workflows that plot the file as lines or points did not differ, and thus, we have chosen an arbitrary candidate. The new graph corresponds to the tools labelled with **E0**, **E1**, **E2** and **E2'** in Figure 6.1. The output of our workflow is presented in Figure 6.6.

Extension 3: Plotting the elevation dataset

We can observe that the current workflow does not properly utilise the provided elevation data (Figure 6.6). The reason for this is the plotting tool (*psxy_P*) used in the process. The idea of the specified workflow extension was to introduce detailed elevation data and to use a tool that can depict that elevation on the map (i.e. using a rich colouring scale). However, the extended workflow only distinguishes between positive and negative elevations, plotting them as black and blue tiles, respectively. To solve this issue and draw an appropriate relief map, the part of the workflow

ID	Constraints
E4.1	Tool Color_palette_generation should generate output used by **2D_surfaces**
	F (∃ x (<**Color_palette_generation**^{0,1}(x)> F<**2D_surfaces**^{1,0}(x)> true))
E4.2	Use tool **Gradient_generation**
	F <**Gradient_generation**^{0,0}()> true
E4.3	Use tool **Draw_color_range**
	F <**Draw_color_range**^{0,0}()> true

Table 6.7: Final workflow constraints.

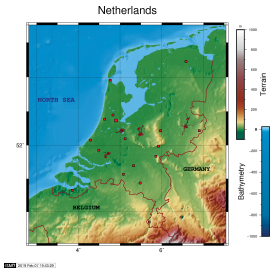


Figure 6.8: Final map.

using the mentioned tool needs to be redefined. We expect the usage of a tool, tailored for plotting surfaces based on elevation data, and thus, our synthesis requires a constraint enforcing it (**E3.1** in Table 6.6).

The synthesis over the two constraints (**E2.1** and **E3.1**) combined with the existing set of constraints results in solutions of length 12. Note that similarly to the initial step, the synthesis in Extension steps 2 and 3 extended our workflow with two new elements each time, even though we have introduced only one constraint per extension. This time, however, the reason for it was the tool annotation part of the domain model, more specifically the dependency between input and output types of the workflow elements.

We chose to run one of the solutions which use the provided XYZ table to generate the required grid file, rather than importing a new one. The generated workflow comprises **E0** - **E3** labelled tools in Figure 6.1). The generated map is presented in Figure 6.7.

Final Extension: Topographical and bathymetrical features

The current map (Figure 6.7) has some inconsistencies with the actual coastline of the Netherlands. We pinpointed Amsterdam's airport (Schiphol) on the map to illustrate the issue. The problem is the elevation of the country, as about one third of the Netherlands lies below sea level and our basic elevation colour palette depicts all negative heights as blue. To solve this issue, we have to separate the plotting of the topographical and bathymetrical features, which requires a further extension of our workflow.

To ensure the desired behaviour we have to specify the appropriate constraints. The plotting tool usage was covered by using the constraint used in the previous step (**E3.1** in Table 6.6). However, this constraint does not guarantee that the tool would be used twice, nor the usage of a different colour palette, which is crucial to distinguish sea and land elevations below 0. As we need a new colour palette, we can be more specific than in the constraint **E3.1**, and say that the initial surface should use the imported colour palette (**E4.0** in Table 6.7), while the second layer of surface should use a new palette (**E4.1** in Table 6.7). Additionally, we would like to emphasise the topographical features of the Netherlands, by shading the generated map (**E4.2**). Finally, we enforce drawing the elevation legend - colour scale (**E4.3**). We combine the four constraints with the constraints used in the previous steps (omitting **E3.1**, as it is captured by **E4.0**) to generate the final solutions.

The first solutions are found at depth 17. Similar to the previous case, the workflows use the appropriate plotting tools, but the solutions differ when it comes to the colour palette data. Some solutions require the file to be imported from the system, while the other generates the colour palette from a part of the already provided colour palette file. As the second option seems more intuitive and does not require us to manually generate another colour palette file, we have selected it (**E4** labelled sub-workflow in Figure 6.1). The new extended workflow is presented in Figure 6.1 with labels **E0**, **E1**, **E2**, **E3** and **E4**. As an illustration, the concrete workflow solution provided in by APE v2 is presented in Figure 6.9. The result of the final extension step is the map in Figure 6.8.

6.1.4 Summary

The case study demonstrates how workflow synthesis technology simplifies the discovery and creation of geographic data manipulation processes once an adequate domain model is available. It proposes an iterative workflow synthesis approach, where the users can obtain desired workflow solutions as a result of small incremental problem specifications. This approach allows users to synthesise large and complex workflows in a few intuitive specification steps.

We designed an illustrative workflow scenario along with the corresponding domain-specific vocabularies and formalised domain knowledge. To the best of our knowledge, no such taxonomies or ontologies and tool annotations are yet available in the geovisualisation domain, thus we contribute an elemental example for the classification of tools and data types. Although it copes well with our use case, it is still not the ideal solution, in particular with regard to scalability to larger application scenarios. Ideally, such domain models would be standardised and defined by the corresponding scientific communities and provided in a structural way, analogously to the CCD ontology of geo-analytical concepts (presented in Section 6.2), or the EDAM ontology and bio.tools registry in the bioinformatics domain (discussed in Section 6.3).

6.2 Geo-Analytical Question Answering

This study uses a scenario with typical geo-analytical questions that can be handled by a GIS, to evaluate the SLTL^x-based workflow synthesis quality over a well-annotated domain. In addition, it evaluates the benefits of modelling geo-analytical concepts as combinations of multiple (four) data dimensions. Finally, this section provides a specification example that utilises the data-tool dependencies, to improve further the synthesised solutions.

The study revolves around livability in Amsterdam, and it uses openly available[6] data from the city of Amsterdam and comparable sources. The general task is to derive livability indicators for elderly people for each postcode area on level 4 (PC4) in Amsterdam, using different urban environmental factors which make the area livable for the elderly. Coping with the diversity of these factors makes the scenario

[6] `https://data.amsterdam.nl`

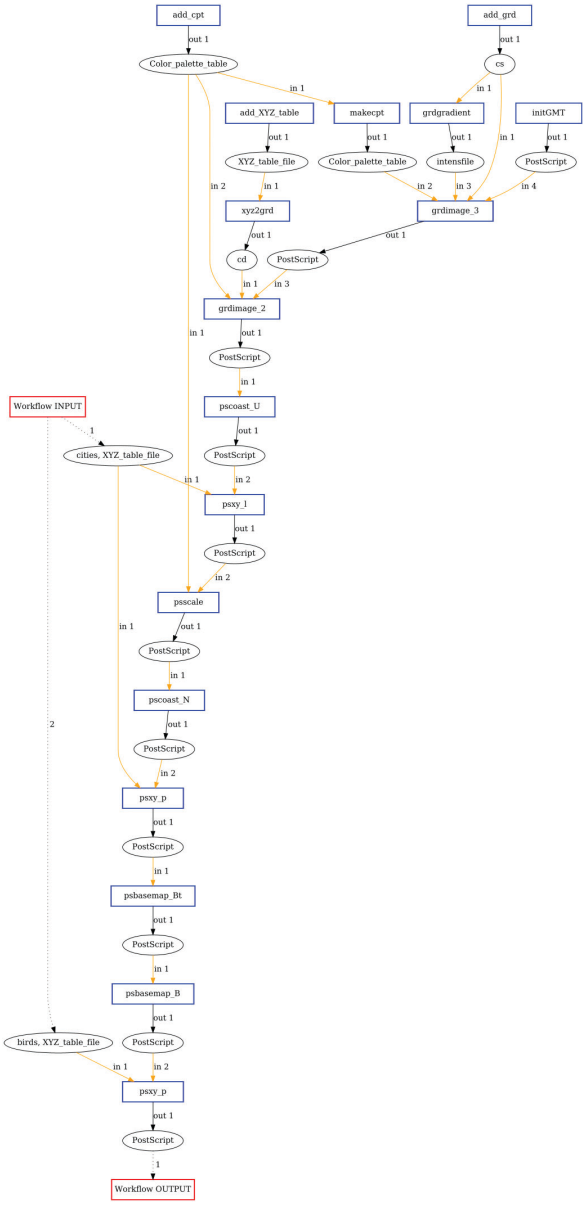


Figure 6.9: A workflow solution generated by APE v2 for the final (E4) specification

challenging. For this study, we formulated five questions as detailed below[7].

The case study introduces two semantic domain models. The first model comprises semantic tool annotations based on the proposed Core Concept Data types (CCD) ontology [120]. CCD defines data types as intersections of OWL classes representing combinations of geo-analytical concepts from the following four disjoint semantic data dimensions: (1) *geometric layer types*, which generalize geometric properties of layers, (2) *core concepts* of spatial information [85], which capture what these layers represent. (3) *measurement levels* of attributes, as well as the notion of (4) *extensiveness*. The second model is used as a benchmark and comprises semantic tool annotations based on an existing benchmark ontology of terms (BDT). The case study uses five geo-analytical questions to assess the relation between the quality of workflow synthesis results and the utilised domain model.

All data used to run the case study and generate the results is available at `https://github.com/quangis/gis_workflow_generation/tree/61452627c50bb89e73df8759088ac06ffb6ae033`. Furthermore, the scripts used to process the semantic domain annotations are available at `https://github.com/simonscheider/SemanticPipelines/tree/cf3c5af3a0114cf502fceee1e0578127e2e8cdf2`.

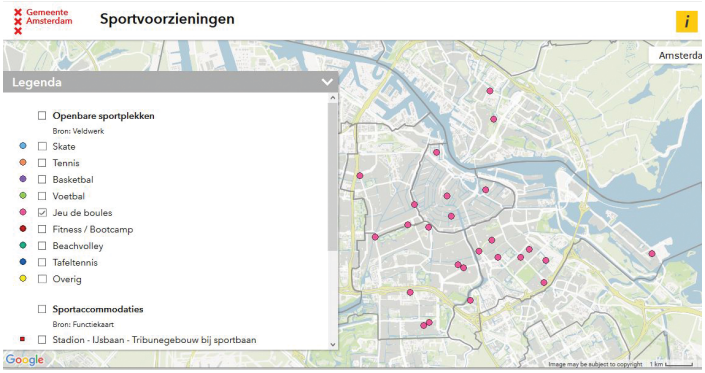
6.2.1 Problem Description

This section introduces five geo-analytical models used for the evaluation. For each of the questions, we give a short motivation and specify the geo-analytical tasks that are used in the evaluation. Each task involves (1) extracting goal concepts, i.e., workflow outputs, (2) choosing datasets for generating answers in terms of start concepts, i.e., workflow inputs, and (3) identifying operation constraints, i.e., SLTL^x constraints that enforce usage of specific tools of format "Use operation *Op*" ("$F < Op^{0,0}() > true$" in SLTL^x), were used whenever the question included hints to corresponding functions. Our specifications using the Core Concept Data types (CCD) ontology are listed in Table 6.8. The BDT version of these specifications in the same table corresponds to the benchmark ontology, a taxonomy of common GIS types (elaborated in the following section).

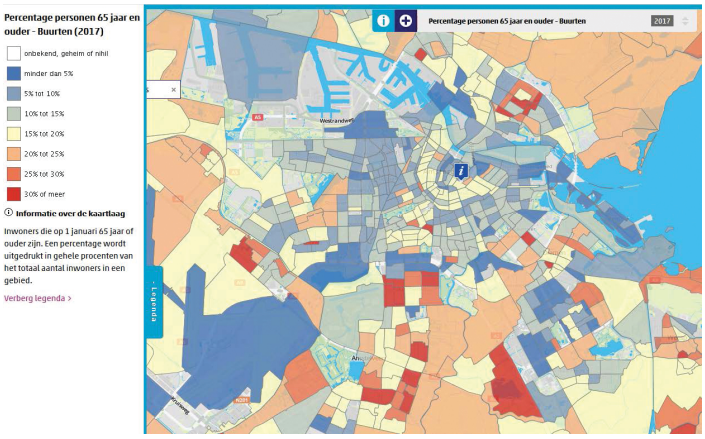
1. **What is the number of sports facilities in each PC4 area?**
   Motivation: Elderly people might prefer particular facilities, such as places for playing Pétanque or Boule.
   Given data (input): Sports facilities (Figure 6.10a) are interpreted as objects, and represented by point vectors with a nominal attribute denoting the facility type; PC4 areas[8] in Amsterdam form a Vector Lattice.
   Goal specification (output): The goal is a Vector lattice on the PC 4 level with extensive counts.
2. **What is the proportion of elderly people living in each PC4 area in Amsterdam?**
   Motivation: Elderly people may prefer neighbourhoods where they can meet

[7] Note that practically assessing livability would require more indicators, such as walkability, crowding and social security. For our purpose, the questions, however, cover a set of sufficiently different concepts.

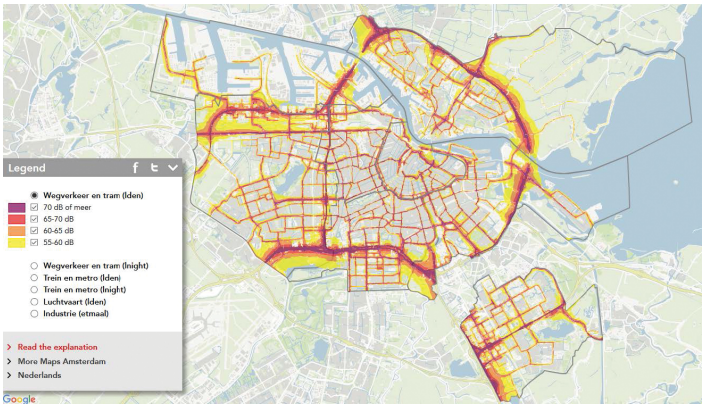
[8] PC4 areas are administrative regions sharing the first * digits of their postal code. They are always used as input data in the following and therefore are only mentioned once.



(a) Sport facilities (Jeu de Boule) in Amsterdam. This helps to answer Question 1.



(b) CBS Buurt statistics, showing the percentage of persons over 65 in neighbourhoods. This answers Question 2.



(c) Noise map of the Amsterdam Municipality, with intervals given in dB. This can help answering Question 4.

Figure 6.10: Map data sources to assess liveability, taken from `https://maps.amsterdam.nl/open_geodata`.

peers, or may conversely be happy to live in an area with many young people.
Given data (input): The CBS Buurt statistics (Figure 6.10b) contains percentages of elderly in the population of a neighbourhood. It is interpreted as a Vector Lattice with a ratio-scaled (intensive) attribute.
Goal specification (output): The goal is a ratio-scaled intensive attribute of a Vector Lattice on the PC 4 level.

3. **What is the accessibility of parks for each PC4 area in Amsterdam?**
Motivation: Elderly people might prefer living in neighbourhoods where parks are within reach so they can easily take a walk.
Given data (input): The CBS land use dataset (BBG)[9] can be used to select areas with parks. It is interpreted as a Coverage with nominal attribute denoting the land use type ("Park en Plantsoen").
Goal specification (output): The goal is a ratio-scaled attribute of a Vector Lattice on the PC 4 level.
Operation constraints: The term "accessibility" in the question implies an answer which makes use of some distance measurement.
4. **What is the amount of noise pollution in each PC4 area in Amsterdam?**
Motivation: Elderly people might prefer living in neighbourhoods where there is a low amount of noise.
Given data (input): The map of traffic noise levels (Figure 6.10c) is interpreted as a Contour map with an ordinal attribute denoting the noise interval in dB.
Goal specification (output): The goal is an ordinal scaled attribute of a Vector Lattice on the PC 4 level.
Operation constraints: The term "amount" implies aggregating the noise field over the PC4 area. Therefore, we added the constraint that some aggregation method, like zonal aggregation, should be used.
5. **What is the average temperature within each PC4 area in Amsterdam?**
Motivation: Elderly people are especially sensitive to urban heat islands, so they might prefer neighbourhoods with low average/maximum temperature in the summer.
Given data (input): A map of pointwise meteorological measurements[10] with an interval scaled attribute denoting temperature.
Goal specification (output): The goal is an interval scaled attribute of a Vector Lattice on the PC 4 level.
Operation constraints: As above, the term "average" implies aggregating the temperature field over the PC4 area. Therefore, we added the constraint that some aggregation method, like zonal aggregation, should be used.

As one can see in these examples, to specify the problem we rely on the information given in the questions as well as the information about available data sources to the largest possible extent. This includes specific semantic interpretations of the sources. Though such interpretations might be done differently in some cases [120], we believe the chosen ones represent a defendable expert view of the analytic tasks.

[9] `https://www.cbs.nl/nl-nl/dossier/nederland-regionaal/geografische-data/natuur-en-milieu/bestand-bodemgebruik`

[10] For example, as provided e.g. by the KNMI at `http://www.klimaatscenarios.nl/toekomstig_weer/transformatie/index.html`.

Question	Ontology version	Input specification	Output specification	Constraint
What is the number of sport facilities in each PC4 area?	CCD	ObjectPoint⊓NominalA Lattice⊓ VectorA	CountA⊓VectorA⊓Lattice⊓ERA	
	BDT	PointA VectorRegionA	VectorRegionA	
What is the proportion of elderly people living in each PC4 area?	CCD	Lattice⊓VectorA⊓RatioA⊓IRA Lattice⊓VectorA	Lattice⊓VectorA⊓RatioA⊓IRA	
	BDT	VectorRegionA VectorRegionA	VectorRegionA	
What is the accessibility of parks for each PC4 area in Amsterdam?	CCD	Coverage⊓NominalA Lattice⊓VectorA	RatioA⊓VectorA⊓Lattice	*Use operation* Distance
	BDT	VectorRegionA VectorRegionA	VectorRegionA	*Use operation* Distance
What is the amount of noise pollution in each PC4 area in Amsterdam?	CCD	Contour Lattice⊓VectorA	OrdinalA⊓VectorA⊓Lattice	*Use operation* ZonalStatistics
	BDT	VectorRegionA VectorRegionA	VectorRegionA	*Use operation* ZonalStatistics
What was the average temperature within each PC4 area?	CCD	IntervalA⊓PointMeasures Lattice⊓VectorA	IntervalA⊓ VectorA⊓Lattice	*Use operation* ZonalStatistics
	BDT	PointA VectorRegionA	VectorRegionA	*Use operation* ZonalStatistics

Table 6.8: The questions and their inputs, outputs (goals) and constraint specifications in the CCD and the benchmark (BDT) ontology.

Note that, though their specifics are given, solving these analytic tasks still involves nontrivial expert knowledge. For example, based on reading Question 5, a layman might believe one could simply "average" the given point-wise temperature measurements, while the task actually requires estimating and summarising a field. The former would result in a semantic error, rendering the workflow meaningless and therefore useless for the purpose. We test whether semantic annotations provided by CCD can add this level of expert knowledge to the synthesis process.

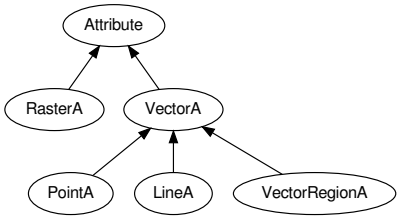
6.2.2 Evaluation Criteria

In general, workflows can be evaluated at *design-time* or *run-time*. Whether a workflow is actually executable can only be evaluated at run-time and involves automatic deployment. However, even if a workflow is readily executable, it still might generate meaningless results that do not answer the question. For this reason, we are more interested in assessing the *meaningfulness* of an answer [118], and this can already be done at design-time using expert assessments. This section explains our framework for doing this, and the results are discussed in Sect. 6.2.3.

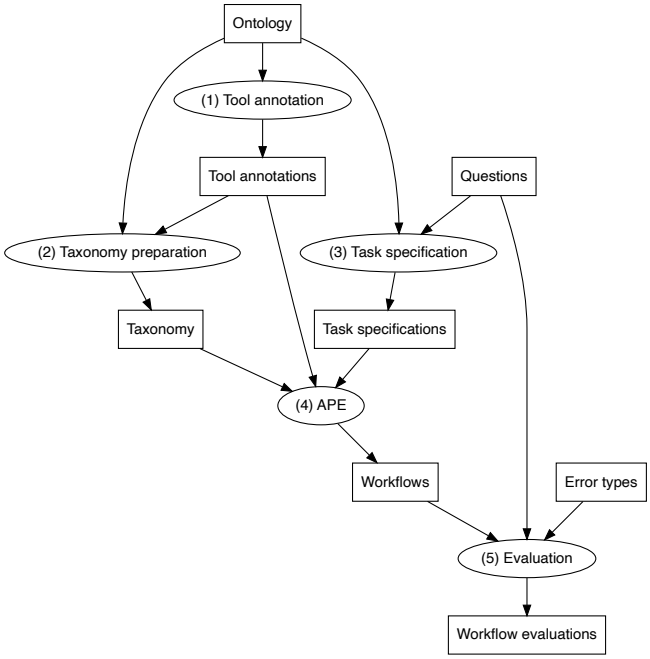
The evaluation of an ontology for workflow synthesis consists of multiple steps (Figure 6.11b) [84]: (1) The tools are annotated with data classes from the ontology. For example, the fact that *Kriging* interpolation transforms a PointMeasure dataset into a FieldRaster dataset is annotated here. (2) In a taxonomy preparation phase, the ontology and the annotated tools are used to create a *taxonomy* of types. This taxonomy is an RDFS hierarchy (consisting of *rdfs:subClassOf* triples) of data and tool classes. (3) Analytical questions are coded into problem specifications, consisting of the provided data types (inputs), goal (desired output type), and constraints over operations. (4) The taxonomy, tool annotations, and problem specifications are fed into APE, which generates a set of up to n distinct workflow solutions up to length k for each specification. In this study, n was set to 20 and k to 8, a longer workflows. (5) The quality of the solutions is evaluated by a GIS expert using the error classification scheme explained in Sect. 6.2.2 and a solid understanding of the questions.

Synthesis Evaluation Criteira

To evaluate SLTL^x-based synthesis using the CCD ontology, we compared the synthesised workflows against workflows obtained when using a benchmarking ontology. The goal is to measure the improvement that conceptual/semantic types add to workflow synthesis. The benchmark should reflect the types provided by current geodata structures. More precisely, we generated a subset of CCD where all conceptual dimensions (including core concepts and measurement levels) were removed and which only includes one semantic dimension related to geometry types, namely the distinction between raster and vector attributes, as well as between point, line and region attributes (see Figure 6.11a). Note that the class Tessellation was also removed since it does not occur in current data structures. We call this ontology the *benchmark data types* (BDT) ontology. Using this simple ontology, we manually created corresponding tool annotations by substituting every type with the least upper bound (supremum) concept that is still in BDT. In the same way, we generated BDT versions of the problem specifications, as listed in Table 6.8.



(a) The *benchmark data types (BDT)* ontology only consists of the well-known raster and vector data types, and a further specification of the different vector types. It is a subset of the CCD ontology defined on the attribute level.



(b) A summary of our ontology evaluation framework for workflow synthesis. For an ontology, five steps are performed (see text for explanations). All steps are done both for the CCD ontology and the benchmark ontology as a benchmark to measure improvements.

Figure 6.11: Elements of our evaluation framework.

Error Types and Precision Measures

We use a quality assessment approach from information retrieval [19] to evaluate the GIS workflow quality. The idea is that workflow synthesis is treated like a retrieval process, and its precision is measured by the extent to which the synthesised workflows answer the given question. In principle, one could measure both precision (the proportion of retrieved answers that are correct given all retrieved answers) as well as recall (the proportion of retrieved answers that are correct given all correct answers), however, the latter is difficult since it requires a complete and correct answer set generated by experts. Another problem is the definition of correctness in terms of error types. The quality of GIS workflows is evaluated using a schema of four error types on two different severity levels, which are explained below, and summarised and illustrated in Table 6.9.

Error severity	Error type	Example workflows
Hard	Signature	Figure 6.12c
	Semantic imprecision	Figures 6.12a, 6.12b
Soft	Redundancy	Figure 6.13b
	Data quality	Figure 6.13c

Table 6.9: An overview of the different error types.

Hard errors are critical errors which result either in a wrong or non-meaningful answer, or in a workflow that is non-executable due to wrong data formats. Correspondingly, we distinguish two kinds of hard errors: *signature* errors, which have a part of the workflow that can not be executed because a tool is incorrectly applied, and *semantic imprecision* errors, which produce a meaningless or invalid answer for the given question, because the ontology misses some required semantic constraint of applicability of data, tools or some information contained in the question.

Soft errors are non-critical errors where workflows *do* entail a correct answer, but which are in some sense of lesser quality. We distinguish two kinds of soft errors: *redundancy* errors, where workflows make use of tool applications which are unnecessary for giving a valid answer, and *data quality* errors, where workflows contain transformations that diminish the *geodata quality* of the result in a way which is unnecessary, but which still render the workflow useful for the task. Geodata quality has many dimensions, among others, positional and attributes accuracy, granularity/precision (≈ resolution) and completeness [54]. Geodata quality comes in degrees, so geodata is never perfectly accurate, precise, and complete. Furthermore, data transformations never increase the quality and GIS workflows usually entail some quality loss. In our case, quality errors mostly included unnecessary reductions of the *spatial resolution*, e.g., based on applying unnecessary focal statistics which tends to blur a raster. For example, the workflow in Figure 6.13c shows how an interpolated raster is blurred in this way before being aggregated with zonal statistics.

Figure 6.12 illustrates three hard errors. For all of these, the answer either is not meaningful for the given question, or the workflow is not even executable. The workflow in Figure 6.12a is supposed to answer Question 3 about the accessibility of

parks, and it was generated based on CCD. It converts landuse polygons to a landuse raster, and subsequently counts the variety of landuse types in a neighbourhood around each raster cell. This 'landuse diversity' is subsequently reclassified to an existence raster (e.g. by selecting a certain range; this is unspecified in our tools). The next operation calculates the Euclidean distance to this filtered landuse diversity. Finally, the average distance to the filtered landuse diversity is computed in each PC4 area. Clearly, this workflow is not meaningful for our question, and it is hard to imagine a scenario where it would be. For this reason, it is classified as a semantic imprecision error.

The workflow in Figure 6.12b is also supposed to answer Question 3 and was generated based on BDT. It uses the landuse dataset *directly* for distance measurement, and therefore the resulting raster represents the distance to *any* landuse polygon. Because the landuse polygons cover the entire extent, this will always be 0, and is therefore not meaningful, and thus classified as a semantic imprecision error. A signature error occurs in Figure 6.12c, which is supposed to answer Question 1 about the number of sports facilities. Here, the points with sports facilities, which have a nominal attribute that indicates the facility type, are *summed* in every PC4 area. Nominal attributes are usually encoded with strings, but numbers are expected. For this reason, the workflow is not executable, and it is classified as a signature error.

6.2.3 Results

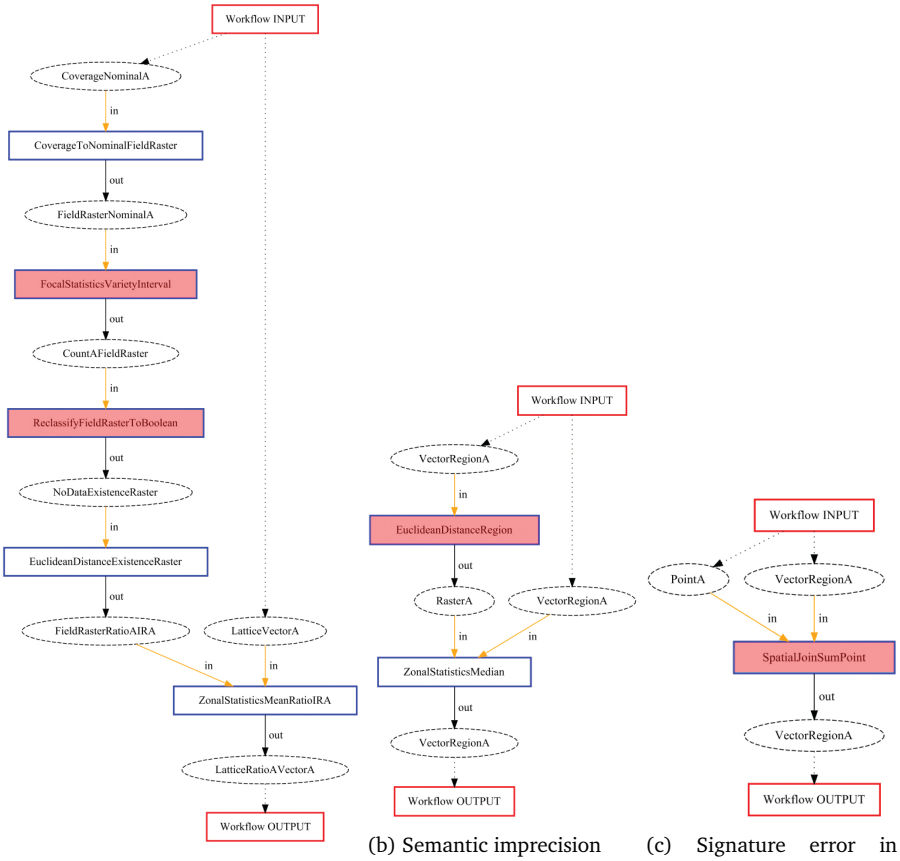
To assess the value of the CCD ontology relative to the benchmark, we evaluated synthesised workflows in the manner described in the last section. In this section, we report on the results and discuss their implications.

Synthesis Evaluation

We counted errors for workflows that were synthesised with both ontologies in the study (Table 6.10). In this table, we report soft errors only for those workflows that did not have any hard errors. Thus, the sum of hard and soft errors can be at most equal to the number of workflows. The column "correct" shows the number of workflows without any hard errors.

Table 6.10 shows the evaluation results of the study, including 172 workflows in total[11]. These are less than 200 due to more restrictive ontological constraints, which often prevented APE from reaching the maximum of 20 workflows. We can see that the hard error rate falls from 86% down to below 1% for CCD. Vice versa, this means that while only 14% of all BDT workflows were correct and answered the questions, almost 99% of all CCD workflows were meaningful answers to the posed questions. This gap can be directly attributed to the missing semantics in BDT. It is also interesting to see that BDT provoked 17% signature errors, because common geodata types do not include information about certain attribute value types that are important for syntax errors. It is also apparent that redundancy errors in the main study are very frequent for CCD, at 70% of all correct workflows.

[11] https://github.com/quangis/gis_workflow_generation/tree/61452627c50bb89e73df8759088ac06ffb6ae033/evaluation

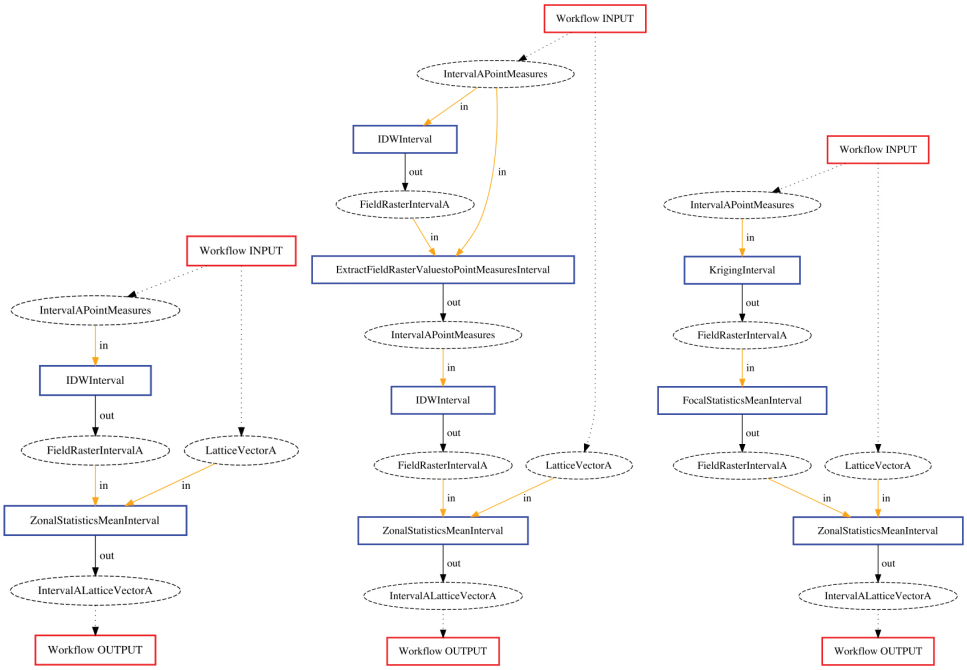


(a) Semantic imprecision error in workflow for Question 3 (CCD).

(b) Semantic imprecision error in workflow for Question 3 (BDT).

(c) Signature error in workflow for Question 1 (BDT).

Figure 6.12: Examples of hard errors for workflows synthesised for questions 3 and 1. Erroneous function applications are highlighted in red.



(a) Correct and no soft errors. (b) Correct, but redundancy error. (c) Correct, but data quality error.

Figure 6.13: Examples of different soft error types for workflows synthesised for Question 5 ("What is the average temperature within each PC4 area in Amsterdam?") using the CCD ontology.

Question	Ontology	Workflows	Correct	Hard errors		Soft errors	
				Signature	Semantic imprecision	Redundancy	Data quality
1	BDT	20	2	13	18	3	0
1	CCD	20	20	0	0	19	3
2	BDT	20	2	4	18	0	0
2	CCD	8	8	0	0	7	0
3	BDT	20	2	0	18	0	2
3	CCD	20	19	0	1	7	6
4	BDT	20	3	0	17	0	2
4	CCD	4	4	0	0	3	0
5	BDT	20	5	0	15	0	3
5	CCD	20	20	0	0	14	12
1-5	BDT	100	**14** (14%)	17	86	3 (21%)	7 (50%)
1-5	CCD	72	**71** (99%)	0	1	50 (70%)	21 (30%)

Table 6.10: Table shows a breakdown of the errors for the main study with questions and ontologies. It also includes tool constraints.

We suggest that CCD causes more redundant workflows compared to BDT, precisely because it enforces more restrictive conditions on workflow synthesis. In consequence, the only way to produce longer workflows is to concatenate redundant tools. This explanation is also consistent with another observation, namely that the number of workflows CCD produced is often lower than the upper limit - 20 (see questions 2 and 4), showing that the space of possibilities of reaching the goal is very limited. In other words, lower amounts of hard errors and higher redundancy/limited workflow diversity turn out to be two sides of the same coin. This becomes more clear when looking at example workflows.

Furthermore, it is also interesting that the data quality error, though reduced by CCD, is still rather high in all cases, showing that a sufficient constraint on geodata quality is not captured by our ontology. This is not surprising since a data quality specific constraint was not included in the synthesis specifications. For example, though two workflows both may aggregate data, the one producing a dataset of higher resolution might be preferable because positional uncertainty is reduced. Yet, such measures and corresponding constraints require a different approach and are considered future work.

To better understand these results, we illustrate the workflows created by APE and their quality for Question 5: "What is the average temperature within each PC4 area in Amsterdam?". Figure 6.13a shows a workflow which is a near perfect answer to the question: It takes the temperature measurements and performs inverse distance weighted interpolation to produce an interpolated temperature field raster. Subsequently, with zonal statistics, it uses the temperature field and the PC4 areas to compute the mean temperature in every PC4 area.

Figure 6.13b shows a different workflow with exactly the same result. The redundant part of this workflow starts when the temperature field raster is converted to temperature point measurements. The resulting IntervalAPointMeasures data object is (for all intents and purposes) *exactly* the same as the IntervalAPointMeasures object that was provided as input. This is because the interpolated field is equal to the interpolated points' values at the points' locations, and exactly those locations are extracted from the field. After this redundant part, the workflow proceeds to calculate the correct answer as in Figure 6.13a.

A more serious quality error occurs in Figure 6.13c. Here, the temperature field is blurred, because the application of FocalStatisticMeanInterval computes the mean of the temperatures within a radius of each raster cell. After this operation, the workflow calculates the answer in the same way as Figure 6.13a, but the resolution of the answer is decreased. Apart from these soft errors, concatenations and combinations of redundant and data quality errors also occur, and are also classified as soft errors.

Extending the Operation Constraints

The original case study [84] relied on a simple SLTL specification, derived from the user questions. Out of the 5 questions, 2 did not specify any semantic constraints, apart from the specified inputs and outputs. The remaining 3 scenarios specified a constraint each, of the form *"Use operation X"*. Based on the limited specification and the observation presented in Section 6.3, it is expected that some of the solutions

include redundant steps.

We identify a rule that geo-analytical workflows should follow, to omit the redundant transformations detected so far. Namely, data objects should not be transformed more than once, as consecutive transformations include redundant steps in a workflow. At the time of the writing of the case study, SLTL, the underlying formalism behind the APE v1 framework did not support such a constraint format. The expressive power of SLTL^x, i.e., the formalism behind APE v2, however, allows us to introduce the rule as an additional constraint on the domain model level, i.e., a constraint that should be satisfied by each workflow within the domain. We specify the rule in the SLTL^x logic as follows:

$$\begin{aligned}\Phi_2 = &\neg(\boldsymbol{F}\exists x_1(< Transform^{0,1}(x_1) > \boldsymbol{F} < Transform^{1,0}(x_1) > true))\wedge \\ &\neg(\boldsymbol{F}\exists x_1(< Transform^{1,0}(x_1) > \boldsymbol{F} < Transform^{1,0}(x_1) > true))\end{aligned} \tag{6.1}$$

As presented in the evaluation section of Chapter 3, using the additional constraint allows the APE framework to exclude approximately 70% of the detected soft errors, labelled as *redundancy errors*. The constraint demonstrates the importance of data-tool dependency constraints. Using data-tool dependency to accurately specify the problem, provides substantially improved synthesis results.

6.2.4 Discussion

Our results demonstrate that workflow synthesis with core concept data types as semantic constraints enables us to automate the design of GIS workflows for a diverse set of geo-analytical tasks on a high-quality level. This means that hard errors which would render the workflow useless for the purpose seem to be almost entirely prevented, given that input data of the right purpose and quality is available. Furthermore, the four semantic dimensions used to describe geo-analytical concepts, and accurate domain descriptions that utilise the SLTL^x formalism, e.g., specifying data-tool dependencies, can be used to further improve the solutions and omit the majority of sub-optimal solutions.

The presented results have several important implications:

1. It indicates that common geo-analytical *questions and tasks* might translate well to loose specifications using SLTL^x and the CCD ontology. Tasks including accessibility assessment, spatial interpolation and summary statistics can be specified using core concepts, measurement levels, as well as constraints over a semantic hierarchy of tool concepts.
2. It indicates that the CCD ontology might provide a solid *semantic basis for annotating GIS functions and data*, and for constraining their application to ones that are meaningful under the given task. This issue is not obvious, as it is still unknown which semantic level would be needed for geo-analytical purposes.
3. It indicates that our way of *benchmarking and evaluation* based on information retrieval might be used as a general method for quantifying the impact of semantic information on geo-analytical task solving. Though semantic background knowledge is known to be important for data analytics [121], it is com-

monly hard to measure its impact on information products. For this reason, ontology engineering often suffers from not being able to show its benefits. Workflow quality benchmarking provides a way to account for this.

4. It indicates that workflow synthesis with CCD could be a way to approach the problem of indirect question-answering (indirect QA) [121]. In indirect QA, questions cannot be answered directly, by retrieval or inference from knowledge bases, but they require adequate transformations. GIS workflows are a very good example of the relevance of such a system since geographic questions are seldom answerable without data transformations.

6.3 Proteomics Data Analysis

This case study uses four proteomics use cases introduced by Palmblad et al. [111] that describe typical scenarios for the analysis of proteomics data. Since the initial case study the domain model, as well as the synthesis framework, have evolved. The bio.tools registry [66] has grown and matured substantially [67], as well as the EDAM ontology of bioinformatics terms [64]. Finally, the PROPHETS framework [108] used for automated workflow exploration in the previous study has been discontinued, with APE v2 taking its place.

We use APE v2 for the workflow exploration and move "into the wild" [76]: Instead of using a small, handcrafted set of tools and annotations, we work with tools and annotations directly from the bio.tools registry. This opens up an unprecedented wealth of tools and accordingly a huge number of possible alternative workflows. With APE we systematically explore the space of possibilities. However, as the quality of semantic annotations in bio.tools varies, deviations in the quality of the automatically explored workflows are to be expected. Therefore, we focus on evaluating the quality of the suggested workflows.Our goal is to evaluate the synthesis quality over large real-life domains, and ultimately, to assess APE as an "off-the-shelf" synthesiser.

The remainder of this section is structured as follows. First, we describe the experimental setup. Then we present and discuss the results from the workflow exploration experiments, before concluding the section with a short summary.

The data resources and code for running this study, along with the workflow exploration results and evaluation data, are available online at `https://github.com/sanctuuary/Proteomics_domain_setup` and in the Supporting Information[12] for [76].

6.3.1 Problem Description and the Setup

Here we describe the setup of the study, summarised in Figure 6.14, where APE is the workflow exploration tool used. This includes (1) the process of fetching and filtering the semantic tool annotations from the bio.tools registry, (2) the workflow use cases and corresponding workflow specifications, (3) the parameters and configurations of the different workflow exploration runs, and (4) the workflow evaluation process.

[12] `https://pubs.acs.org/doi/10.1021/acs.jproteome.0c00983`

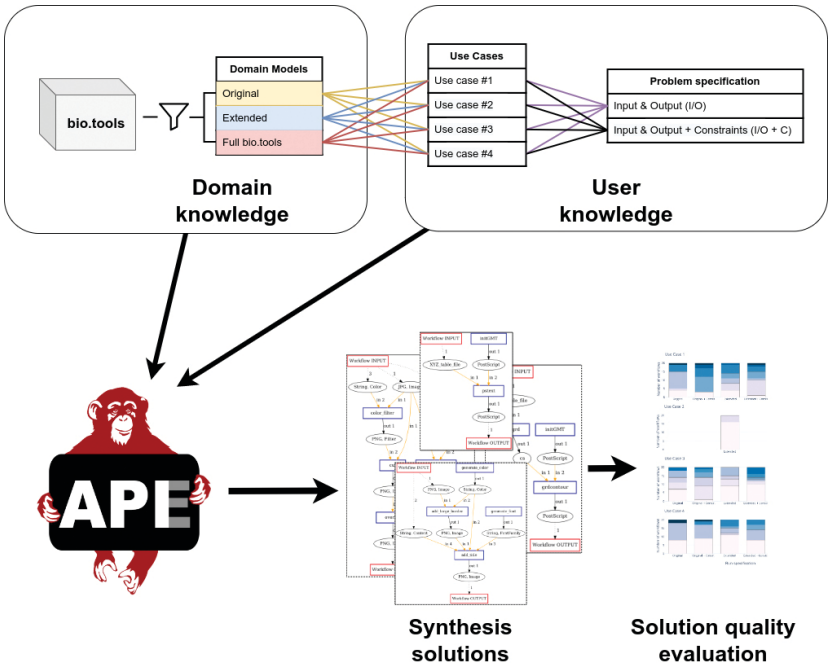


Figure 6.14: Experimental setup of the study.

As discussed in the earlier chapters, APE uses input/output-annotated tools in combination with type and operation taxonomies as a domain model. This approach fully aligns with the information that is available through bio.tools and EDAM.

Concretely, the semantic domain model we provided to APE consists of operation, type and format taxonomies as controlled vocabularies for the description of computational tools (directly derived from EDAM) and functional tool annotations (inputs, outputs, operations performed) using terms from these taxonomies (directly derived from bio.tools). The domain setup process is described in further detail below. The workflow specifications we provide to APE comprise the available inputs and intended outputs (data type and format, again in EDAM terms) and in some cases, additional constraints, provided as SLTL^x NL templates. APE is then used to synthesise the shortest workflows that satisfy the specification.

Domain Models

In the previous study [111], the domain model comprised 26 tools (mostly but not exclusively from the `ms-utils.org` collection of mass spectrometry data analysis tools). The tools were annotated in a CSV file with input and output data types and formats, as well as operations using terms from the EDAM ontology. In this study, we fetched the corresponding tool annotations directly from the bio.tools registry via its REST API. The bio.tools annotation schema also uses the EDAM ontology as reference vocabulary, but in contrast to the previous tabular annotation format, it allows for the annotation of multiple inputs and outputs per tool. Thus we now work with

significantly more comprehensive descriptions of the available tools' functionality.

For this study, we worked with three different domain models (the mentioned number of tools per domain includes only the well-annotated tools from the repository, as described later in the sections):

1. The **original** set of 21 (out of 24) tools (*comet, enrichnet, genetrail, genetrail2, gprofile_r, idconvert, isobar, libra, msconvert, mzXMLplot, Pep3D, PeptideProphet, peptideshaker, proteinprophet, protk, PTMProphet, ssrcalc, searchgui, Tandem2XML, rt, xml2tsv, xtandem, extract_protein_names*), comprising the tools from the previous study that are available in bio.tools.
2. An **extended** set of 271 (out of 751) proteomics tools, corresponding to the labelled proteomics domain in bio.tools (`https://proteomics.bio.tools`), which extends the original set of tools.
3. The **full** bio.tools set of 1,642 (out of 17,369) tools, containing all the well-annotated tools available in the registry.

To create the three domain models, we (1) used the bio.tools REST API to fetch the JSON files containing the respective bio.tools native annotations (which can contain multiple function annotations per tool), (2) cleaned the set of annotations, by keeping only well-annotated functions (details follow below), and (3) transformed the annotations to the APE annotation format. The quality of the domain model determines the quality of the workflows obtained through automated exploration. The bio.tools repository contains thousands of tools, annotated by a diverse group of contributors. Not surprisingly, this leads to mixed levels of annotation quality. In particular, many of the tool annotations lack input and/or output definitions, specify them vaguely or incompletely, or use outdated EDAM references. Therefore, we discarded annotations that:

- do not specify an input, as these tools can be used at any step of the workflow and typically introduce unnecessary new data,
- do not specify an output (typically these are interactive tools), as they do not contribute to solving a data analysis problem,
- miss a data type or format specification, as these are incomplete annotations, or
- reference deprecated EDAM data type or format terms, as they are not part of the domain taxonomy anymore.

Note that there are also many tool annotations that reference deprecated EDAM operation terms. However, we classify those as non-critical annotation errors and allow such tools. The only way that such a domain model could produce wrong results is by restricting usage of the missing tool types through explicit constraints, which we rarely do in the studied use cases.

Table 6.11 summarises the effects of cleaning the annotation sets while creating the three domain models. The annotation quality of the small, original tool set is again good, only three tools were discarded. In the larger, community-curated domain of proteomics we find around a third of the tools to be in a well-annotated format, while on the global level, less than 10% of bio.tools are directly suitable for automated exploration. Still, these new domain models with 271 and 1,642 tools, respectively, provide a next-level challenge for automated workflow exploration and resemble the variety of tools in a real-world setting better than the original set and

Number of	Original	Extended	Full bio.tools
..tools in bio.tools	24	751	17,369
..functions annotated in the tool set	24	858	18,408
..discarded functions	3	587	16,778
..resulting APE annotations	21	271	1,642

Table 6.11: Effects of cleaning the tool annotation sets.

those of the other case studies.

Workflow Specification Use Cases

We reuse the four proteomics data analysis use cases from the previous study [111], which require workflows of increasing complexity:

1. Extraction of the *Amino acid index (hydropathy)* from peptides in biological sample measured by liquid chromatography MS.
2. *Protein identification* and pathway *enrichment analysis* of *MS spectra*.
3. The identification and localization of post-translational modifications in a phosphoproteomic study.
4. Protein quantitation of multiple biological samples labelled by *iTRAQ*.

Table 6.12 summarises the corresponding workflow specifications, which are the input for APE. As EDAM and bio.tools evolved since the previous case study, we revised the workflow specifications to match the now available terms. Concretely, we generalised the output specifications in the first and the fourth use case from the expected data types to their parent classes, as none of the annotated tools used the originally specified types as input/output anymore. In the first use case we substituted *Amino acid index (hydropathy)* with *Amino acid property*, while in the fourth use case we replaced *Gene expression profile* by *Expression data*. Similarly, we updated some of the workflow constraints. The term *validation of peptide-spectrum matches* became obsolete with EDAM 1.19, so instead we opted for the similar *Target-Decoy* term. Further, we updated the term *gene-set enrichment analysis* to *enrichment analysis*. Although it is not deprecated, it is not used in any of the tool annotations.

Workflow Exploration Runs

As illustrated in Figure 6.14 (top), this study comprises 24 different workflow exploration runs in total: For each of the three domain models (Original, Extended and Full bio.tools), we let APE explore possible workflows for all the four use cases. Furthermore, we apply two different versions of the workflow specifications: desired input/output only (I/O) and I/O with additional constraints (I/O+C). This distinction allows us to evaluate the effects of these additional constraints in comparison to only I/O specifications when exploring workflows in large collections of tools. In the previous case study, even with a small set of tools, constraints were crucial to guide the exploration towards the intended workflows, as an I/O specification alone did not provide sufficient information. We expected this to be even more needed with the increased size of the domain model.

We limit the exploration runs to the first 20 (shortest) workflows. This choice is motivated by two observations: First, as workflow candidates get longer, they tend to

Use Case	Inputs	Outputs	Constraints
# 1	*Mass spectra* in *Thermo RAW* format	*Amino acid property* in any format	(i) Use operation *peptide identification* (ii) Use operation *Target-Decoy* (iii) Use operation *retention time prediction* (iv) Do not use operation *Protein identification*
# 2	*Mass spectra* in *Thermo RAW* format	*Pathway or network* in any format	(i) Use operation *peptide identification* operation (ii) Use operation *enrichment analysis* (iii) Use operation *enrichment analysis* only after *peptide identification* operation (iv) Use tool *ProteinProphet* only after *PeptideProphet*
# 3	*Mass spectra* in *Thermo RAW* format	*Protein identification* in any format	(i) Use operation *PTM identification* (ii) Use operation *PTM identification* only after operation *Target-Decoy* (iii) Use operation *Target-Decoy* only after operation *peptide database search* (iv) Do not use operation *Target-Decoy* more than once.
# 4	*Mass spectra* in *Thermo RAW* format	*Expression data* in any format	(i) Use operation *iTRAQ* (ii) Use operation *iTRAQ* only after operation *Target-Decoy*

Table 6.12: Workflow specification for the four use cases.

extend already considered shorter solutions and introduce redundant steps. Second, users of automated workflow exploration tools are able to process and compare a limited number of workflows, and, in our experience, 20 is a reasonable bound in practice. Finally, due to a resource limitation, we terminate the search if no solutions are found until length 20 (workflows with 20 operation steps).

The experiments are run using bio.tools annotations as of November 25, 2020, EDAM version 1.24, and APE version 2.0.1[13] on an i7-6500U CPU at 2.50GHz and 16GB RAM machine running on Ubuntu 20.04. The runtime behaviour of the workflow exploration algorithm is discussed in detail in Chapter 7. For the smallest domain model, the runtime was a few seconds, and exploration using the largest domain model was averaging around 10 minutes, the longest almost an hour. Generally, runtime performance increases with the size of the domain model and the length of the solutions.

Workflow Evaluation

To evaluate the quality of the suggested workflows, two domain experts (proteomics researchers with extensive tool and workflow experience) scored the first 20 workflow candidates of each exploration run on a scale from 0-3 according to the following criteria:

3: good workflow, have seen it or similar before, I know it will work
2: interesting suggestion, seems viable, could work, worth trying
1: might work, but does not seem very useful, or has unnecessary steps
0: I know that it will not work

The two experts scored the workflows independently. We calculated the averages of their scores for subsequent analysis. Further evaluation through implementation, execution and benchmarking, like performed in the previous study [111] was out of scope for this study and is left for future work.

6.3.2 Synthesis Results

In the following, we summarise and discuss the workflows found in all exploration settings described above. We use the number of obtained workflows and the scores they received from the domain experts (see Figure 6.15) as indicators of workflow exploration comprehensiveness and quality in the different setups.

As general observations, the workflow-evaluating domain experts remarked that they found this an interesting and insightful exercise. On the one hand, they came across several interesting, sometimes even surprising, workflow suggestions that they would not have thought about themselves, but that seem worth trying. On the other hand, for faulty or insensible workflow suggestions they could usually see how the (flawed) annotations of the involved tools set the automated composer on a wrong track.

Use Case #1

The workflows we obtained for the first use case with the original tool set closely resemble the corresponding results from the previous study. First workflows for the

[13]The evaluation that this section is based on [76] is performed using an earlier APE version - 1.1.2. However, as the version already included prototypes of the new (SLTL^x) formalism, the results using APE version 2.0.0 remain unchanged.

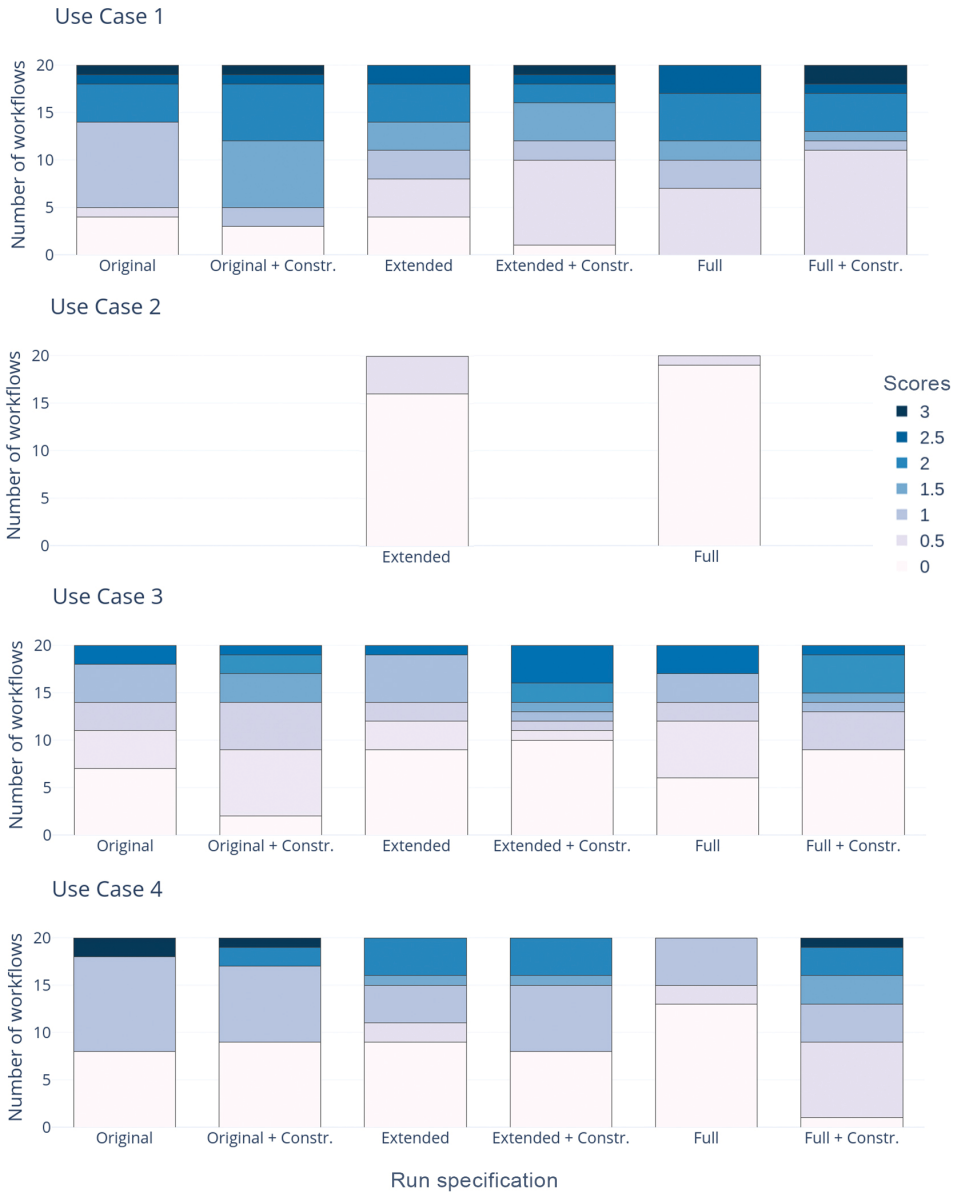


Figure 6.15: Workflow quality evaluation.

Listing 6.1: Selected workflow candidates: Use Case #1

```
Original (I/O)
msConvert->Comet->rt
msConvert->Comet->xml2tsv->rt
msConvert->msConvert->Comet->rt
msConvert->Comet->PeptideProphet->rt
...

Original (I/O + C)
msConvert->Comet->PeptideProphet->rt
msConvert->Comet->PeptideProphet->idconvert->rt
msConvert->Comet->PeptideProphet->xml2tsv->rt
...

Extended/Full (I/O)
DeconTools->Mascot Server->rt
PEAKS De Novo->PeptideProphet->rt
ReAdW->Comet->rt
msConvert->MassWiz->rt
...

Extended (I/O + C)
msConvert->MassWiz->rt
MZmine -> Mascot Server -> rt
msConvert->collect_mgf->MassWiz->rt
mzBruker->mzXML2Search->MassWiz->rt
CompassXPort->Comet->PeptideProphet->rt
...

Full (I/O + C)
msConvert->MassWiz->rt
MZmine -> Mascot Server -> rt
ThermoRawFileParser->MassWiz->rt
ThermoRawFileParser->Comet->PeptideProphet->rt
CompassXPort->DTA to MGF File Converter->MassWiz->rt
...
```

specification are found at a length of three, which corresponds to workflows of three successive tools. As the examples in Listing 6.1 indicate, the main difference from the results of the previous study is that some tools that were previously used (e.g., *X!Tandem*, *SSRCalc*) are not included anymore. This is due to different annotations of the tools, which are both now annotated to expect two inputs instead of one.

The example workflows for the original tool set also show another frequently observed pattern: Short workflows with a conversion step (such as *msConvert*) are often contained within longer workflows which extend it by repeating the conversion step. These redundant steps can be avoided by introducing constraints that prevent multiple conversion operations over the same data. Currently, this requires to formulate such constraints per individual tool, so we are considering ways of adding it as a more convenient configuration option to APE.

When we explore workflows that satisfy the exact specification but comprising the extended and full sets of tools, the number of results increases significantly. As Listing 6.1 indicates, the additional results include several new and sometimes surprising workflow suggestions, such as the combination of using *MZmine* before *Mascot Server*. However, there are also workflows suggestions that are less sensible. For example, some workflows start with the tool *CompassXPort*, although it cannot read *Thermo RAW* files. The reason is that *CompassXPort* is annotated in bio.tools to read files *Mass spectrometry data format*, which is the parent term for 30 specific file formats including *Thermo RAW*. *CompassXPort* cannot read *Thermo RAW*, but this can not be inferred when using general annotations including parent terms. Ideally, *CompassXPort* would be annotated in bio.tools with a precise list of accepted input formats and not their parent term. To work around this with the current version of APE, a constraint can be added that, for example, excludes *CompassXPort* from the exploration. Alternatively, one could restrict the domain model to include only tools described by sufficiently specific file formats. However, given the current state of tool annotations, such a restriction is likely to strongly decrease the domain model size. To solve such problems more generally, we are currently investigating possibilities for extending APE with new configuration options and/or heuristics for data format handling.

Interestingly, we obtained almost the same results for the extended and full domain model. The only notable difference is the occurrence of a rather new parsing tool *ThermoRawFileParser* in the workflow solutions with the full domain model. *ThermoRawFileParser* is not yet included in the proteomics domain as it has been added recently. With the exception of this tool, extending the domain model with tools from outside the bio.tools proteomics domain does not create new possibilities for this use case. This can be interpreted as evidence that the coverage of proteomics tools in the respective bio.tools domain is comprehensive.

A general observation from the evaluation is that the constraints do indeed have a considerable effect on the number and quality of workflows obtained. As Figure 6.15 shows, for all domain models the domain experts gave higher scores to the workflows explored with constraints. This is especially important with the extended and full domain models. There the number of solutions exceeds the threshold of 20 already at length 4 in the unconstrained case. To a large extent, these are workflows that are (presumably) implementable based on their input/output annotations, but

Listing 6.2: Selected workflow candidates: Use Case #2

```
Extended/Full bio.tools (I/O)
ProCoNA
unfinnigan->ProCoNA
msConvert->ProCoNA
OpenSWATH->ProCoNA
...
```

that do not perform the operations actually intended by the workflow developer. Constraints that specify these intentions better thus help to drastically decrease the number of unfeasible workflows. Furthermore, they allow for the exploration of longer and at the same time more meaningful workflows. For this use case the hand-curated domain model appears to be more restrictive and effective for finding appropriate solutions than the constrained case over a larger domain model. We attribute this to the specific tailoring of the original domain model to this use case, an approach that does however hardly scale in practice. This reemphasises the importance of using appropriate constraints when dealing with larger collections of tools and annotations from community repositories.

Use Case #2

When exploring workflows for the second use case with the original tool set, somewhat surprisingly we did not find any. The reason is that the *enrichment analysis* tools (*gProfiler*, *EnrichNet*, etc.), which are needed to generate the specified workflow output, are annotated to expect a *Gene ID* type as one of the two obligatory inputs. However, there is no tool in the domain model that would generate a *Gene ID* as output, hence there is a missing link that prevents the exploration from finding corresponding workflows. A "shim" tool that converts IDs into each other (e.g. *UniProt protein accession* into *Gene IDs*) could provide this missing link. It is an ongoing discussion, however, if such shims should be documented in bio.tools, or if that would lead to an overload of tools with insignificant functionality. Many shims are in fact included in larger software suites or libraries, though not annotated as individual functions of these, and thus missed by automated exploration. Related issues are incomplete annotations due to large numbers of functions performed or formats supported. For example, some enrichment tools accept tens of different ID types but only list a few in their tool annotation. Thus some possible matches are missed. However, using a more abstract parent term in the annotation can cause erroneous matches, as described for *CompassXPort* above.

Such missing links are typical risks of using small domain models. Interestingly, here the use of the larger sets of tools does not resolve the issue. The exploration with the extended domain model does return possible workflows, but these are questionable. As Listing 6.2 shows, the first suggestion is a single tool (*ProCoNa*) that matches the input/output specification. Like *CompassXPort*, it is annotated as using the general *Mass spectrometry data format* as input, while it actually only accepts some of these formats (a table containing peptide identifications), so the problem is

Listing 6.3: Selected workflow candidates: Use Case #3

```
Original (I/O)
msConvert->Comet
msConvert->Comet->idconvert
msConvert->msConvert->Comet
msConvert->Comet->PTMProphet
...

Original (I/O + C)
msConvert->Comet->PeptideProphet->PTMProphet
msConvert->msConvert->Comet->PeptideProphet->PTMProphet
msConvert->Comet->PeptideProphet->PTMProphet->ProteinProphet
msConvert->Comet->PeptideProphet->PTMProphet->idconvert
...

Extended/Full bio.tools (I/O)
PEAKS De Novo
msConvert->ProMEX protein mass spectral library
mzBruker->Comet
ProSight PTM->ProSight PTM
...

Extended/Full bio.tools (I/O + C)
msConvert->Comet->PeptideProphet->PTMProphet
unfinnigan->Comet->PeptideProphet->PTMProphet
T2D converter->Comet->PeptideProphet->PTMProphet
CompassXPort->Comet->PeptideProphet->PTMProphet
...
```

similar to the one described above. The other suggestions are then actually meaningless extensions of this first workflow. In the constrained case, no workflows are returned, as the constraints try to enforce the aforementioned *enrichment analysis* tools, which cannot be used due to the missing shims.

The unconstrained exploration with the full domain model yields the same results as with the extended model, with the exception of the *ThermoRawFileParser* tool for the initial file conversion. As in the previous constrained cases, the constrained exploration with the full domain model resulted in no solutions. Unfortunately, in the case of the full model, we could not explore solutions up to length 20. Due to the exponential runtime complexity and memory requirements of the exploration algorithm, composition at lengths longer than 8 exceeded the available memory, so the exploration process stopped there. However, based on the results from the extended model, we assumed that no solutions would have been found among workflows longer than 8, either.

Use Case #3

For this use case, the workflows obtained with the original tool set largely cor-

respond to the results from the previous study, similar to Use Case #2. As Figure 6.15 shows, there is a notable difference between the runs with and without constraints. In fact, for this use case, the constraints are crucial, as the input/output specification is quite general and does not provide sufficient information to solve the problem as intended. As Listing 6.3 shows, the workflows obtained with the unconstrained specification are in principle valid but lack the validation part of the reference workflow scenarios. Only the use of constraints ensures that *Target-Decoy* tools like *PeptideProhphet* and *PTMProphet* are included.

This observation also holds for the extended and full tool sets. Looking at the workflow candidates, we see that the slight difference in results between the extended and the full domain model is in fact again caused by the additional tool *ThermoRawFileParser* in the full domain. This aligns with our findings from Use Case #1.

Note that for this use case we again observe the spurious use of *CompassXPort*, *T2D converter* and similar tools, for the same reasons as discussed above.

Use Case #4

Similar to Use Cases #1 and #3, the workflows obtained for the fourth use case (see Listing 6.4) highly correspond to those from the previous study when exploring workflows for the original tool set. Furthermore, this scenario affirms again that the usage of additional constraints to specify the problem decreases the number and increases the quality of the workflows (see Figure 6.15). Interestingly, while the constrained solutions over the extended and the full domain differ in the usage of the aforementioned *ThermoRawFileParser* tool, the unconstrained solutions differ in two more tools, namely *TDimpute* and *pyQms*. These two tools, similarly to *ThermoRawFileParser*, have not yet been added to the proteomics domain, but in fact contain the EDAM topic *Proteomics*.

Listing 6.4: Selected workflow candidates: Use Case #4

```
Original (I/O)
msConvert->mzXMLplot
msConvert->msConvert->mzXMLplot
msConvert->Comet->Libra
msConvert->msConvert->Comet->Libra
...

Original (I/O + C)
msConvert->Comet->PeptideProphet->Libra
msConvert->msConvert->Comet->PeptideProphet->Libra
msConvert->Comet->PeptideProphet->PeptideProphet->Libra
msConvert->Comet->ProteinProphet->PeptideProphet->Libra
...

Extended (I/O)
MapQuant
msConvert->MapQuant
MassWolf->mzXMLplot
...
```

```
Extended (I/O + C)
PEAKS De Novo->PeptideProphet->Libra
msConvert->MassWiz->PEAKS Q
unfinnigan->Comet->PeptideProphet->Multi-Q
PEAKS De Novo->CompassXPort->PeptideProphet->Multi-Q
...

Full bio.tools (I/O)
MapQuant
msConvert->MapQuant
MassWolf->mzXMLplot
msConvert->pyQms
MZmine->TDimpute
...

Full bio.tools (I/O + C)
PEAKS De Novo->PeptideProphet->Libra
ThermoRawFileParser->MassWiz->PEAKS Q
msConvert->MassWiz->PEAKS Q
PEAKS De Novo->CompassXPort->PeptideProphet->Multi-Q
...
```

6.3.3 Summary

Annotating computational tools with ontologically defined terms describing their operations, data types and formats enables their automated composition into tentatively viable workflows. Providing such annotations (using terms from the EDAM ontology) is one of the main goals of the bio.tools registry, which has become the principal catalogue of computational tools in the life sciences. We applied the Automated Pipeline Explorer (APE) to the bio.tools registry, revisiting workflow use cases from an earlier proof-of-concept study in proteomics.

Our results show that APE can be successfully used as an "off-the-shelf" synthesiser in such a domain, as the required semantic annotations fully align with the information that is available through bio.tools and EDAM. Furthermore, we demonstrate that this approach can compose purpose-specific workflows of high quality in extensive collections of semantically annotated tools.

6.4 Geo-Event Question Answering

In this case study [79], we focus on a specific type of geo-analytical question that has not been a focus of previous studies: geo-event questions. As it is an ongoing study in its preliminary stages, this section presents only its preliminary findings. Geo-events are most succinctly defined as "something that happens" [43]. This study presents a prototype process for generating workflows to answer geo-event questions. First, we provide annotations of the domain, comprising a tool taxonomy we created from descriptions of geo-operations, a data type ontology obtained

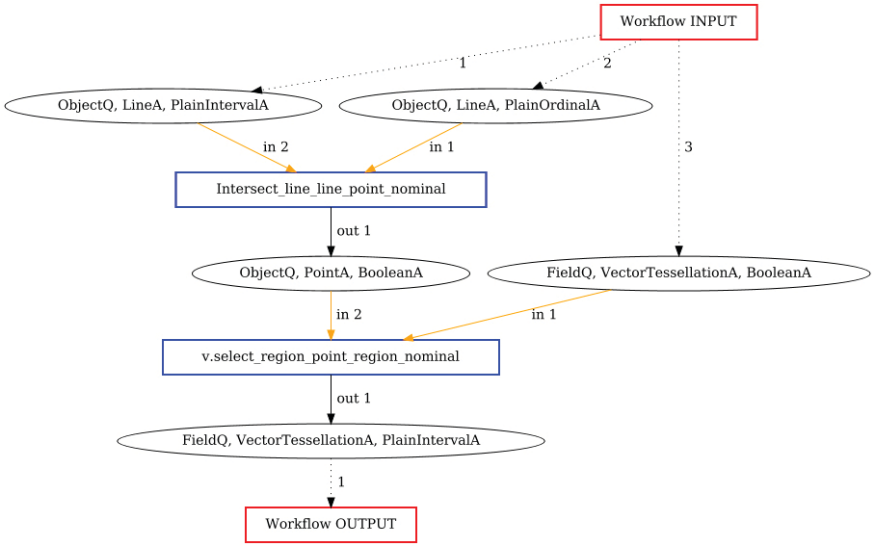


Figure 6.16: A correct workflow solution provided by APE v2

from the Core Concept Data types (CCD) ontology [84], and the annotations of the mentioned geo-operations with respect to the input/output pairs. We created our taxonomy of tools based on Brauners geo-operator categories [21] to include more information about the tools than just data type. In addition, we utilise the CCD ontology for creating the data type taxonomy and for describing data types. Second, we utilise the process of automated composition of workflows (using APE) for the specific geo-event question. Finally, the generated workflows are post-processed to restrict the solution space and provide more structured solutions.

All the resources required for running the APE framework (including the taxonomies and tool annotations) and the post-processing steps can be found at `https://github.com/MohammadUT/Geo_event-QA`.

The preliminary problem specifications comprise solely the input and output specifications, i.e., it does not include any additional temporal constraints. We investigate the first 50 solutions in our runs which domain experts individually evaluate. APE generated 35 workflows (70%) that return the correct answer (e.g., the workflow presented in Figure 6.16). Six answers (12%) are invalid, while nine solutions (18%) are close to the actual answer but do not entirely match it (e.g., they provide a fragment of the correct answer presented in Figure 6.16). This diversity in the quality of the solutions is caused by the similarity of the operation signatures, i.e., similarities between the input and output types.

In the current study we present two different post-processing approaches for grouping equivalent workflows: *intensional* and *extensional*. Their goal is to reduce the solution space and provide more structured solutions. The intensional approach groups equivalent workflows whose tool steps are semantically equivalent (i.e., equivalent in query intentions). The extensional approach refers to group-

ing equivalent workflows that return the same outputs (i.e., query extensions) by running the input data through the workflows and comparing their output results. The main difference between the extensional and intensional approaches is that we might have workflows with different tools that are not semantically equivalent but return the same outputs.

For the 50 generated solutions obtained from APE, the proposed intentional approach restricts the number of solutions to 24 groups. Therefore, 41% of the workflows are subsumed by the corresponding equivalence groups. In addition, the results show that the extensional approach could restrict the 50 APE-generated solutions to only five groups.

Although the preliminary specification shows promising results, we could further improve the quality of the solutions generated by APE by providing appropriate SLTL^x constraints. However, automation of such constraints for any given geo-event question is not trivial and is left for future consideration. Finally, the two post-processing steps could support the exploration of a much higher number of solutions, as the grouping could allow their manual evaluation (e.g., the extensional method restricted the number of solutions to about 90%).

6.5 Discussion

The chapter introduces four case studies from geo- and life-sciences used to evaluate the APE v2 framework, as well as the underlying SLTL^x-based synthesis approach. The following sections provide our observations from those case studies.

6.5.1 Geo-science Domain

While the vision of an entirely automated GIS still appears far-fetched, we believe our geo-science studies show that automatic recommendations of geo-analytical workflows for properly specified goals are within reach. In particular, they can support geoinformaticians and GIS analysts who develop workflows by systematically exploring the space of possibilities with the available tools. Though specifications often need to be formulated and answer workflows still need to be checked (and implemented) by human experts, our approach scales up the geo-analytical process by automatically assessing the potential of a given tool resource for a task, which does not seem possible to date. Our preliminary Geo-Event Study [79] shows a way toward geocomputational code generation and question-answering. A parallel opportunity is implied by the fact that CCD types describe geodata sources. Though the geodata retrieval problem admittedly involves more specific information about geographic phenomena than what is captured by CCD, it might add to the effectiveness of current geographic information retrieval strategies [71].

Another question concerns the *completeness* of the CCD ontology concepts concerning geo-analytical tasks. Which concepts are we lacking and which are relevant for modelling some form of geospatial analysis? Are the current four semantic dimensions sufficient? This is probably not the case, as our workflow evaluation shows. For example, to capture certain functional constraints, such as “Distance”, we need to be able to generalise over corresponding GIS tools. For this purpose,

our collaborators are currently working on a *transformation language for geospatial information*.

6.5.2 Life Sciences Domain

Naturally, the overall quality of the automatically obtained workflows highly depends on the quality of the semantic domain model, in our case the domain ontology (EDAM), and the tool annotations (bio.tools). Despite the volume and maturity of the domain model, many tools in the community-driven bio.tools registry are not accurately annotated. Presumably due to a lack of awareness and understanding among curators of what constitutes good annotations for the purpose of automated tool composition and workflow exploration. Even partly inadequate annotations can lead to wrong compositions or exclusion of tools that suit a requested combination of input, output and operation. Hence, to use bio.tools with APE, the relevant tool sets have to be pre-processed and filtered to sort out poorly annotated annotations. Conversely, small curated domains can yield high-quality results, but they tend to overfit. They do not enable the exploration of new, possibly better-performing tools and workflows and furthermore hardly scale, so they are not an ideal solution in the long term.

Domain models and problem specifications are never expected to be perfect. Therefore, APE's ability to adjust the workflow exploration according to new incremental constraints gains importance. Such constraints are crucial for filtering out nonsensical and undesirable alternatives, and for guiding the search towards actually desirable tool combinations in a still huge space of possibilities, as shown in the geovisualisation case study. The resulting workflows also contain valuable information that can be employed to improve tool annotations. A knowledgeable researcher can adapt the annotations of neglected tools and correct erroneous annotations in tools that were assigned to a workflow despite their inability to fulfil that particular task.

CHAPTER 7

Evaluation

Abstract - This chapter provides a usability assessment of the contributions provided in the dissertation. The usability is assessed based on (1) the runtime performance of the implementation, (2) results from existing applications of APE, and (3) user experiences of the framework. First, the chapter evaluates the runtime performance of APE v2 within the case studies presented in Chapter 6. It identifies the NL templates provided by APE v2 as an important factor that improved the overall encoding runtime. In addition, the chapter proposes a caching function which would further improve the encoding runtime. Second, the three existing case studies are used to demonstrate the applicability of APE v2. Finally, the results from the survey of APE users help us assess concrete features of the framework.

This chapter evaluated the usability of the introducedSLTL^x-based approach and the APE v2 framework. Section 7.1 assesses the runtime performance of APE v2 within the case studies presented in Chapter 6. It presents the *encoding* and the (SAT) *solving* runtime within the framework. In addition, the section identifies improvements in the encoding size and runtime when using the predefined SLTL^x natural language (NL) templates. As described in Chapter 4, these templates are commonly used SLTL^x constraints provided in a natural language. They provide an abstraction over the SLTL^x language as well as optimisation steps within the propositional problem encoding. Section 7.2 presents three additional applications of APE, used to illustrate the applicability of the SLTL^x-based synthesis approach. Section 7.3 presents the results from a survey of researchers and students that used APE in their research. The goal of the survey is to assess the usability of the APE framework from the perspective of the users.

7.1 Runtime Performance of APE v2

APE v2 relies on the SAT-based encoding and the significant progress in the development of SAT-solving heuristics. The heuristics have evolved to yield highly efficient SAT solving, e.g., unit propagation, clause learning, back jumping, etc. [99]. Furthermore, SAT solving continues to improve at an impressive pace, driven by an annual competition [42].

The satisfactory runtime benefits of the SAT solving technique are noticeable when executing the case studies presented in this chapter (The runtime results for each case study introduced in Chapter 6 are presented in Table 7.1 (rows 1-9). The experiments were performed on a PC with a 2.50GHz i7-6500U CPU with 16GB RAM running on Ubuntu 20.04. The recorded times were recorded as average runtimes out of 10 individual runs. We have used APE v2.0.2 over the semantically annotated domain models provided at `https://github.com/sanctuuary/APE_UseCases`.

The APE runtime in the presented case studies can vary substantially depending on the size of the domain as well as the size of the solutions. Relatively smaller to medium workflows (up to length 10), as well as domains up to 100 tools, in practice result in an average runtime of few seconds (see rows 1-3, 6 and 7 in Table 7.1). Long workflows (over length 13), as well as medium-sized domains (comprising a few hundred tools), result in runtime between 20 and 60 seconds (see rows 4,5 and 8 in Table 7.1). Large domain models (comprising a few thousand tools) result in a runtime of up to an hour (see row 9 in Table 7.1).

The efficiency of SAT solving runtime is reflected in the APE v2's *solving* runtime. In practice, the solving time takes approximately between 6% and 20% of the total APE v2 runtime. The *encoding* runtime, however, presents a bottleneck in our algorithm. It takes approximately 60% to 80% of the presented APE execution times. The main reason is the exponential blowup of space with respect to the length of the solutions/size of the domain[1]. Therefore, we distinguish two main challenges in our approach, (1) *problem encoding*, i.e., parsing and encoding complex SLTL^x formu-

[1]Size of the domain is reflected in the size of the domain taxonomies, tool annotations as well as domain constraints

las, and (2) *domain encoding*, i.e., encoding large workflows and/or large domain models. Although the state explosion cannot be avoided, we try to mitigate it. This section discusses the two challenges, and proposes approaches that aim to improve the corresponding encoding runtime and/or size.

Problem Encoding

Problem encoding refers to a propositional encoding of a specific problem, i.e., input, output and used constraints. APE v2 supports the specification of arbitrary SLTL^x formulas when defining a synthesis problem. Chapter 4 introduces the transformation of an arbitrary SLTL^x formula into a propositional encoding. The transformation uses recursive methods to encode the **G**, **F** and **U** modal operators. As such, the complexity of a formula, i.e., nesting of modal operators and variable quantifiers, is directly related to the size of the propositional encoding. Table 7.2 illustrates the dependency between the encoding runtime/size and the problem specifications. It presents examples from Geovisualisation [73] (GMT, rows 1-9) and Geo-Analytical Concepts [84] (G-A, rows 9-11) case studies.

A natural language SLTL^x template[2] (listed in Chapter 4), on the other hand, has a predefined structure which allows for optimisations of the corresponding propositional encoding. Utilising optimisation techniques presented in Chapter 4) NL templates reduce the encoding runtime and size when compared to the equivalent constraints specified directly in SLTL^x.

The optimisation depends on the complexity of the formulas. Table 7.2 compares the runtime of problem encoding within the presented scenarios. It compares the runtime of problems specified using closed-text templates, and problems that specify some (or all) of the constraints in SLTL^x. Rows 3 - 6 show the encoding size and runtime when solving **E1** (1st extended workflow) from the Geovisualisation case study. Rows 3 and 4 compare the runtime when 3 of the problem constraints (**E1.4**-**E1.6**) are specified in SLTL^x, while row 5 and 6 show the synthesis results when 6 of the constrains (**E1.1**-**E1.3** and **E1.4**-**E1.6**) are specified in SLTL^x. All of the mentioned constraints are of the form "$F <$ *Op*$^{0,1}() >$ *true*" that corresponds to "Use operation *Op* in the solution" template. We see that when using simple constraints (that do not nest modal operators) the encoding size and runtime do not differ between the two types of constraint specifications (e.g., row 5) even when we are exploring longer workflows (e.g., row 6). For example, when searching for workflows of length 15, the difference between the size of the encodings, i.e., number of clauses, is around 20%, while the runtime is approximately 10% longer. We have similar results when looking at the examples from the Geo-Analytical Concepts study, disregarding the workflow length. However, complex SLTL^x formulas, i.e., formulas that include nested modal operators and existential quantifiers, drastically increase the encoding size/runtime. When we express **E1.1**-**E1.3** formulas in SLTL^x (in form "$\mathbf{F}(\exists x(< Op_1^{0,1}(x) > \mathbf{F}(< Op_2^{1,0}(x) >$ *true*$))$"), that enforces port binding between operations Op_1 and Op_2, the encoding runtime doubles for workflows of length 10 (see row 1 of Table 7.2). When we extend the search to workflows of length 15, the encoding using APE v2 stops due to a timeout (set to one hour).

[2]APE framework provides a set of frequently used SLTL^x formulas as natural-language templates, to simplify the usage of the system.

	Domain	Use Case	Number of solutions	Lengths explored	Total APE runtime (sec)	Encoding runtime (sec)	SAT solving runtime (sec)
1	Geovisualisation	**E0**	20	1-6	3.08 s	1.87 s (60.71%)	0.59 s (19.16%)
2		**E1**	20	6-10	9.31 s	6.04 s (64.88%)	1.52 s (16.33%)
3		**E2**	20	6-10	7.31 s	4.83 s (66.07%)	1.39 s (19.02%)
4		**E3**	20	10-13	18.04 s	12.44 s (68.96%)	2.64 s (14.63%)
5		**E4**	20	15-17	40.08 s	23.83 s (59.46%)	10.56 s (26.35%)
6	Geo-Analytical Concepts	**Q1**	20	1-8	9.18 s	7.44 s (81.06%)	0.62 s (6.98%)
7	Proteomics (original)	**No1 + C**	20	1-6	9.58 s	7.06 (73.70%)	0.97 s (10.13%)
8	Proteomics (extended)	**No1 + C**	20	1-4	62.13 s	45.89 s (73.86%)	3.737 s (6.01%)
9	Proteomics (full bio.tools)	**No1 + C**	20	1-4	1,217.30 s (22.3min)	829.74 s (68.16%)	
10	Extended Proteomics	**No1**	1	4	25.35 s	19.08 s (75.27%)	2.11 s (8.32%)
11		**No1 + C**	1	4	26.87 s	19.18 s (71.38%)	2.02 s (7.52%)
12			2	4	26.99 s	19.29 s (71.47%)	2.42 s (8.86%)
13			10	4	28.3 s	19.6 s (69.26%)	3.31 s (11.61%)

Table 7.1: Comparison of APE v2 runtime in presented case studies.

	Domain and Use Case		Length	Constraints evaluated	\SLTLx constraints			Natural Language Templates		
					Encoding size	Encoding runtime	SAT runtime	Encoding size	Encoding runtime	SAT runtime
1	GMT	E1	10	E1.1 E1.3	538 805	4.73 s	0.37 s	360 674	2.48 s	0.45 s
2			15	E1.1 E1.3	-	timeout	timeout	1 029 878	7.01 s	1.09 s
3			10	E1.4-E1.6	363 744	2.77 s	0.52 s	360 674	2.48 s	0.45 s
4			15	E1.4-E1.6	1 128 171	7.97 s	1.4 s	1 029 878	7.01 s	1.09 s
5			10	E0.1-E0.3, E1.4-E1.6	367 839	3.01 s	0.53 s	360 674	2.48 s	0.45 s
6			15	E0.1-E0.3, E1.4-E1.6	1 259 250	8.33 s	1.43 s	1 029 878	7.01 s	1.09 s
7		E3	13	E3.1	-	timeout	timeout	706 207	4.71 s	0.69 s
8			15		-	timeout	timeout	1 030 187	7.10 s	1.14 s
9	G-A	Q3	4	all	81 198	1.92 s	0.36 s	77 430	1.21 s	0.3 s
10			10		407 562	3.17 s	1.12 s	350 434	2.58 s	0.95 s
11			15		1 082 385	6.92 s	5.84 s	836 459	6.19 s	4.34 s

Table 7.2: Comparison of APE v2 runtime over user specifications that utilise constraint templates, and those that specify them directly in SLTL^x. Each run was stopped once the first solution was found (timeout was set to one hour). GMT and GA labels are used to represent Geovisualisation and Geo-Analytical Concepts domains, respectively. (* - Size of the encoding is measured in number of clauses used in the CNF form).

Therefore, using the natural language (NL) templates in this scenario is crucial to finding solutions in the set time frame.

The NL templates provide an intuitive interface for the user that is not familiar with the underlying SLTL^x logic. In addition, their usage results in a better overall runtime of the framework. For that reason, our case studies, as well as the results presented in Table 7.1, use the NL templates to define the problem specifications (constraints). Our aim is to further evaluate the usage of constraints in the ongoing (and potentially new) case studies and improve the list of supported templates accordingly. In addition, we actively introduce new constraints based on the feedback from the community, either by directly contacting us or by creating issues at our GitHub repository[3].

Domain Encoding

Domain encoding refers to the propositional encoding of the workflow, as well as the domain model (taxonomies and tool annotations).

The exponential blowup in the search space is inherent to the temporal logic-based synthesis problems. As we consider larger workflows we have to encode exponentially more possible combinations of operations and the corresponding data types. The ratio between the size of a workflow and the corresponding propositional encoding is presented in Table 7.2. We can see that the encoding size, i.e., number of clauses in CNF format, grows with the size of the workflow (e.g., compare rows 1 and 2, or 3 and 4).

We notice, however, that the encoding time does not increase much when we look for more than one solution at a specific length (see rows 11-13 in Table 7.1). The encoding time remains approximately the same regardless of whether we are looking for 1, 2 or 10 solutions[4]. The main reason is that APE v2 generates the encoding to find the first solution and reuses it in the following steps, i.e., when searching for alternative solutions.

Based on the deterministic method used to create the propositional encoding (presented in Chapter 4) we know that the domain encoding remains the same for each problem, as long as the domain model and the workflow length are constant. Therefore, we propose to implement a cache function, that initially (the first time the domain is used) stores the domain encoding and reuses it each time the domain model is reused. This would drastically reduce the APE runtime, as the new encoding step would only propositionalise the problem-specific constraints, while the rest would be retrieved from the cached encodings. To estimate the benefits of the approach, we need to assess the ratio between the domain and problem encoding[5] runtime in our examples.

Row 10 in Table 7.1 shows the runtime of APE v2 when finding a single solution for scenario No1 in the Extended Proteomics domain [76] (see Section 6.3) with no constraints specified. We see that the encoding of the domain (as there were no additional constraints specified) takes 19.09 seconds on average, which is 75% of APE runtime. Row 11 shows the runtime of APE v2 for the same problem, which

[3] `https://github.com/sanctuuary/APE`

[4] As expected, the solving time increases slightly as each solution requires a separate solving step.

[5] Time needed to encode problem-specific constraints, specified using NL templates.

includes temporal constraints (specified as NL templates). We notice that APE v2 takes an additional fraction of a second to encode the domain constraints, and an additional second, in total to solve the problem. The domain encoding still takes 70% of runtime. Similarly, rows 2 and 8 in Table 7.2 show the runtime, as well as the size of the encoding, when solving the first and third extended workflow from the Geovisualisation study [73] (see Section 6.1), respectively. In both examples, we generate workflows of length 15. We notice again that the additional constraints do not increase the runtime drastically when encoding a specific workflow length.

In the presented case studies the domain encoding accounts for approximately 60% – 80% of the APE v2 runtime. Additionally, reusing an existing encoding (e.g., when used by APE to explore multiple solutions, see row 2 in Table 7.1) does not bring a runtime overhead within the current implementation. Therefore, by keeping the domain encoding, we could omit the domain encoding step within the framework and drastically improve the APE runtime.

APE v2 currently keeps internal mapping functions for different encoding steps to shorten the runtime, however, it does not implement caching of the whole encoding files yet. We plan to implement the function in one of the next APE releases.

Performance Summary

The runtime evaluation demonstrates a good average APE runtime in the existing case studies. As APE v2 relies on the SAT-based encoding, it yields highly efficient SAT solving (usually within a few seconds). Encoding of such a problem, however, suffers from the state explosion problem. As such a problem cannot be solved, we propose approaches to mitigate the problem.

The SLTL^x-based approach provides high flexibility of problem specifications, however, complex SLTL^x constructs might drastically increase the runtime. We identify the usage of NL templates, which implement certain encoding optimisations, as crucial for a good runtime performance. To further improve the average runtime in practice, we aim to extend the number of available templates and potentially improve their structure.

The encoding of the domain, i.e., semantic domain annotations and the workflow structure, is another critical aspect of encoding, as it takes on average 60% – 80% of the APE v2 runtime. The domain encoding, however, does not change as long as the domain model is unchanged. Therefore, we plan to implement a caching function, that would store the encoding after the first run of the domain, and reuse it in all the subsequent runs. This approach could improve the runtime performance by the extent of the domain encoding, therefore, up to 60% – 80%.

We hope that the two optimisation techniques will be sufficient to keep the APE v2 runtime low even in new upcoming domains. In addition, improvements to the mechanism that translates SLTL^x formulas into propositional logic (e.g., by implementing parallel computing), could reduce the runtime gap between the NL template specifications and those specified directly in SLTL^x.

7.2 Third-Party Applications of APE v2

This section presents an overview of three recent studies in geo- and life-sciences, which utilise the synthesis approach behind the APE framework. The studies are used to illustrate the usefulness of the SLTL^x-based synthesis approach.

7.2.1 Spatial Network Analysis

In their recent work, Scheider and Jong [119] tackle domain modelling and automated workflow composition (using APE) in the spatial network analysis domain. The domain comprises methods for measuring accessibility potentials and analysing flows over transport networks. The study focuses on the analysis of football clubs and their fans in the Netherlands. They try to automatically compose solutions to 12 different network analysis tasks, such as "What is the potential number of fans within a travel distance for municipalities?". They propose semantic domain annotations in the spatial network analysis domain, and use them to evaluate the synthesis results.

The domain experts evaluated within the study the quality of 181 workflows. Using the APE framework over the proposed semantic domain annotations showed high accuracy of generated solutions. The study demonstrates that when using an accurate domain model, APE can have a recall of 100% of the optimal solutions. Finally, manual evaluation of the synthesised solutions showed that 59% of all the solutions provided by APE over the proposed model were correct (without any semantic or syntactic errors) and could be used.

7.2.2 NL queries for GIS analyses

Recent prototype [122] for natural language (NL) queries for GIS analyses developed at "Disy"[6] (original, "Disy Informationssysteme GmbH") utilised APE for GIS-workflow composition. The goal of the study was to develop a platform that users (non-GIS experts), can use to answer GIS questions, such as, "Where is the closest green area from my location?" or "Where can I go sledging in my city?". To accomplish that, the initial prototype aims to answer a set of example queries.

The approach covers four steps, (1) *query analysis*, where a natural language processing is used to parse the given question, (2) *data selection*, where the system extracts relevant data types (corresponding to the CCD ontology concepts [84]) to form the specification, (3) *generating GIS-workflows* using the APE framework and (4) *visualisation of results*, by executing the workflows. To accomplish the third step of the analysis, i.e., to generate the GIS workflows, the system integrates APE using the provided APE API, described in Chapter 5. The resulting prototype demonstrates the usability of the APE API, as well as the high quality of the generated solutions. The study is a promising step towards using APE as a plug-in for larger platforms, something we envisioned while designing the APE interface.

[6] Disy is a company that develops software and provides consulting in Data Analytics, Reporting and GIS (`https://www.disy.net`).

7.2.3 Amplicon Sequence Data Analysis

Marker gene sequencing is a well-known and widely used approach, affordable to nearly any laboratory, due to a low number of necessary reads per sample. However, the complexity of the analysis process of the obtained results requires computational skills that are often beyond the scope of a current molecular biologist/ecologist. Cascabel Pipeline [1] was developed to address this issue and provide an easy-to-use amplicon sequence data analysis pipeline. The pipeline uses Snakemake [82] to integrate the computational steps. It provides a highly versatile software that allows users customisation at several steps of the pipeline.

We have recently started a collaboration with the authors of the Cascabel Pipeline, aiming to explore potentially new execution paths for the analysis. The main goal is to find alternative steps within the pipeline, by exploring the relevant fragment of bio.tools. Such steps can be used either to verify the existing steps or to be used as alternatives when needed, e.g., when the user has case-specific restrictions that prevent the usage of the original pipeline step.

The efforts so far were focused on identifying relevant fragments of bio.tools and curating the corresponding tool annotations. The preliminary exploration of amplicon sequence data analysis workflows using APE v2 shows promising results. The study generates workflows up to length 20 within the explored solutions that replicate the original pipeline. The goal of the study is to fully replicate the Cascabel Pipeline and in the process detect equally useful pipeline alternatives.

All the data used to run the case study so far and generate the results is available at `https://github.com/sanctuuary/Automated_Cascabel_Pipeline`.

7.3 APE v2 User Experiences

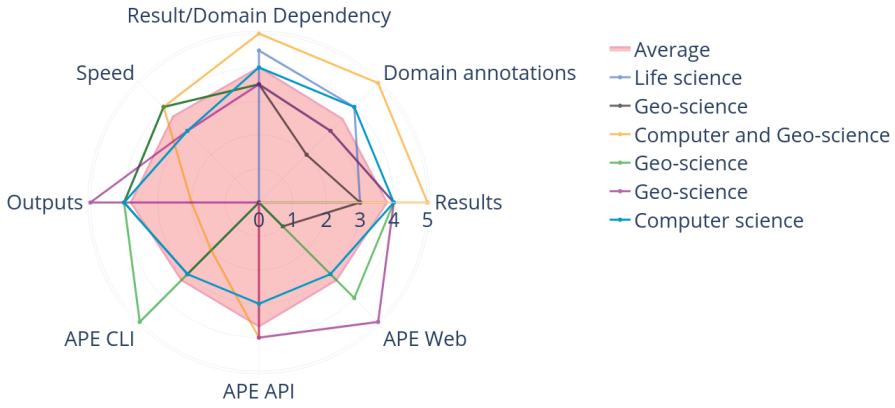


Figure 7.1: Results from the survey of APE users

This section presents results from a survey of six users of the APE framework. The participants were researchers and students from computer-, life-, geo-science domains that used APE in their research. They evaluated the usability of the frame-

work, as well as the quality of the results. Figure 7.1 illustrates accumulative results for the following questions. The average score is highlighted in red, while unanswered/skipped questions are traced as 0 and not included in the average score.

- **Results**
  How would you rate the APE synthesis results in your scientific domain?
- **Domain annotations**
  How would you rate the quality of your domain annotations?
- **Result/Domain Dependency**
  According to you, how much does the quality of the solutions depend on the quality of your domain annotations?
- **Speed**
  How would you rate the speed of the APE synthesis in your scientific domain?
- **Output**
  How would you rate the format of outputs that APE provides (text files and figures)?
- **APE CLI**
  How would you rate the APE command line interface?
- **APE API**
  How would you rate the APE API?
- **APE Web**
  How would you rate APE Web?

The survey included additional open-ended questions[7], as well as 30 minutes follow-up discussion with each of the participants.

The participants agreed that the overall quality of the solutions is of a high level. They discussed flaws in semantic annotations of their respective domains, and identified clear effects they have on the quality of the synthesised solutions. The newly created geo-science domains were rated lower than the more mature life science domain model, comprising the EDAM ontology [67] and the bio.tools registry [66].

The users identified the following features as important. (1) An approach such as the one provided by APE API is crucial for automating question-answering frameworks. Question answering (QA) in GIS is an open problem that requires many components to be fully automated. One of the components requires a synthesis approach that can utilise the existing domain annotations and be easily integrated within a larger QA framework. (2) Setting up a new domain based on the documentation[8] is a straightforward task. In addition, the users acknowledged that APE provides easy integration with existing domain models (in bio- and life-sciences), as well as a repository of existing annotations, that can be used as templates. (3) APE API was well documented and easy to set up. (4) The synthesis process runtime works well in practice. (5) Figures as output give a nice overview of the workflow structure. (6) APE Web can be used to explore new workflows, as well as to test whether the existing domain annotations are suitable for a new case study. For example, when developing the natural language query answering for GIS analyses [122] the scien-

[7] Results of the survey are available at `https://forms.office.com/Pages/AnalysisPage.aspx?AnalyzerToken=MJz6yySn6kQQGhRC67ZOsHGliVk5E2bH&id=oFgn10akD06gqkv5WkoQ5zaY_Ew9ftJBinjp4IZm_YBUQ0ZGVkxGODBSRDdDTjFUMk41RERQUVQyTi4u`.

[8] `https://ape-framework.readthedocs.io/`

tists used APE Web to evaluate the suitability of the existing Geo-Analytical Concepts domain annotations [84] for their purpose.

Based on the survey results, we identified various points of improvement, some validated our existing plans, and some established new future steps. The most notable suggestions are as follows. (1) APE could be integrated with a tool repository such as bio.tools, and provide a quality benchmark, i.e., an evaluation of how well-suited an annotation is for the workflow synthesis. The benchmark would be used to guide the semantic tool annotation process. In addition, it could provide a metric that illustrates how often the tool is utilised in synthesised solutions. (2) Semantic annotations of tools and data types within APE Web could include external references that would be displayed as part of solutions. For example, the user might be interested in a description of a data type, or an external reference to the official website of a tool. (3) Workflow figures could alternatively be provided as *RDF* graphs. (4) APE API could allow domain annotations to be provided as data objects, and provide outputs directly in the *json* format. (5) Users requested additional constraint templates that are currently in process of being implemented.

The current semantic tool annotations (e.g., bio.tools) usually do not account for shims, i.e., scripts used to transform data formats to make two tools compatible. For example, in the proteomics domain, it is often the case that scripting the shims takes as long as setting up a workflow. Therefore, having a repository of available shims would be quite beneficial. The issue with the current repositories is that the shims as such do not qualify as independent tools, and their place within the semantic annotations is not defined. However, even a repository of possible transformations within the domain (e.g., a *CSV* format can be transformed to *TSV* format), without the concrete implementation (scripts) being provided, would be beneficial. It would allow the synthesis framework to discover new workflows that are not possible without the needed format transformations.

Finally, the synthesis approach accepts only parameterised tools as a part of the semantic tool annotations, i.e., the parameters of a tool are always preset for a specific domain. However, in question answering, tool parameters, such as the radius of a map overlay are part of the user-specified question. For example, the question "Which parks are within a 2km radius from a school?" specifies the radius of 2km as a tool parameter. Therefore, if we are to automatically answer such questions, the APE framework has to facilitate such tool parameter specifications. Unfortunately, due to the variety of semantic tool annotations (e.g., APIs, command line tools, web services, etc.), this is not an easy task. One approach would be to restrict the domain, before addressing the issue.

CHAPTER 8

Conclusion

The goal of this dissertation is to provide a solution to scientific workflow synthesis problems that occur in practice. To provide such a solution, this dissertation introduces a temporal logic framework which captures the structure of some of the existing scientific annotations and uses it to synthesise correct-by-construction solutions. Chapter 3 introduces the logical formalism - SLTL^x-based workflow synthesis [72], while Chapters 4 and 5 present the implementation of the synthesis approach - the APE framework [74]. Chapter 6 presents several case studies in geo- [73, 79, 84] and life-sciences [76] that utilise APE v2 and its SLTL^x-based workflow synthesis. The studies illustrate how a new semantic domain annotation can be set up and how existing ones can be used. Each of the case studies shows the benefits of specific features of the SLTL^x-based synthesis approach. Finally, Chapter 7 assesses the usability of the synthesis approach. It demonstrates a promising runtime performance of APE v2, provides three relevant applications of the framework, and presents user experiences obtained as a result of a survey of APE users.

8.1 Outlook

The dissertation demonstrates the usability and versatility of the APE framework and the underlying SLTL^x-based synthesis approach. This section presents an outlook on future developments of the approach. The future directions can be categorised as follows, (1) extending further SLTL^x synthesis problem, (2) improvements of the propositional encoding and runtime, (3) improvements of the APE framework, and (4) improvements of the existing semantic domain annotations.

The presented case studies have not identified substantial limitations in the SLTL^x syntax which reflects on the quality of problem specifications, and in turn on the generated solutions. However, there are some concepts that should be captured accurately. The results of the survey confirmed the need for a repository of *shims* (i.e., for format conversion tools that change the data format but maintain the data type), therefore it would be beneficial if the formalism can distinguish between shims and regular operations on a conceptual level. In addition, it would be useful if the formalism could quantify the taxonomy terms (such as constraints that should hold for all tools in a set individually, e.g. when using constraints to avoid repetition of the same tools).

Due to the low complexity of the bounded approach and its suitability for the existing use cases, we opted for the SLTL^x-based bounded workflow synthesis approach (see Chapter 3). On the other hand, the SLTL^x-based dynamic workflow synthesis supports the synthesis of workflows of arbitrary lengths, at the expense of solving complexity. Such a synthesis approach would be suitable for a framework that compares automatically the composed solutions, and thus the workflow length does not play a role in the evaluation. The automated evaluation and benchmarking are, however, difficult to achieve within the framework, as they require resources that go beyond the domain annotations, such as annotated benchmarking datasets, executable files that correspond to the annotated operations, an execution platform, etc. Developing such a platform is part of an upcoming project, as discussed below. The dynamic synthesis approach could, therefore, be incorporated with such a platform.

Section 7.1 discusses the limitations of the existing transformation from SLTL^x formulas into a propositional encoding. As previously mentioned a caching function would provide a substantial runtime improvement as it would omit the domain encoding. The improvement would be crucial when it comes to larger domains (e.g., proteomics domain). When it comes to problem specification encoding, providing more diverse NL templates, and/or allowing their composition, would decrease the need for constraints specified directly in SLTL^x, and improve the overall runtime performance. However, if we want to have a more substantial specification encoding improvement, the translation algorithm itself should be optimised. As an example, an optimisation could remove unnecessary recursion calls in the case of nested modal operators, or employ parallel computing when transforming multiple similar fragments of the specification. Finally, the solving runtime could potentially be improved by using a different SAT solver. Another option would be to change the reasoning engine to an SMT solver (e.g., Z3), however, our preliminary implementation[1] resulted in a higher runtime. In addition, due to the fact that the implementation of Z3 is not available as a Java library, and has to be installed separately, the resulting APE library lost its "portability".

A web-based graphical interface has a lot of potential for improving the overall automated workflow composition process. At the moment the visual comparison of generated workflows is based on pairwise differences in the structure. The platform could cluster together similar solutions and visualise them. This way, the user would be able to explore more solutions and focus on structures that they find interesting.

Section 5.5 points out that none of the mentioned synthesis approaches provides automated benchmarking of the solutions. Such an approach is, however, crucial for providing production-ready workflows in an automated fashion. This motivated an upcoming collaborative project of international life and computer researchers [92]. The project is supported by the Netherlands eScience Center[2] and will begin in September 2022. It aims to provide a platform that supports automated exploration, implementation, execution and benchmarking in one coherent framework. The resulting platform should assist the workflow developer in systematically exploring and evaluating possible workflows for a specific research question in bioinformatics. The platform aims to combine the APE synthesis approach, used for generating the candidate solutions, with a benchmarking platform based on the OpenEBench [25] platform for community-based benchmarking of bioinformatics resources.

Finally, the heuristic of searching for the shortest solutions provides a workable approach in most cases, however, it is not ideal. The synthesis research community regards particular domain-specific search heuristics (exploiting e.g. non-functional properties or additional knowledge about, for example, the preferred ordering of tools) as crucial towards efficient workflow synthesis in practice [20, 123]. The framework could use the WorkflowHub[3] platform, which has been recently released to host semantically annotated workflows, to assess the non-functional tool properties (e.g., the frequency of tool occurrences) and rank the solutions according to

[1] SMT-based implementation of APE is available at `https://github.com/sanctuuary/APE/tree/ExtendedBitVecImplementation`.

[2] `https://www.esciencecenter.nl/`

[3] `https://workflowhub.eu/`

derived criteria. The approach could later be extended to workflow repositories, such as a GIS repository of expert workflows. The repository is currently being developed as part of the Question-based analysis of Geographic Information with Semantic Queries (QuAnGIS) project[4] and aims to collect workflows of high quality made by domain experts.

8.2 Concluding Remarks

This dissertation tackles the scientific workflow synthesis problem. It introduces the SLTL^x-based synthesis approach to overcome the limitations of the existing SLTL-based formalism. The dissertation introduces a transformation algorithm that translates the SLTL^x specifications into propositional logic. Furthermore, the transformation is implemented as part of the APE framework, which uses the MiniSAT solver [38] to synthesise a solution for the given propositional encoding.

The availability of APE as a concrete scientific workflow synthesis tool allowed for collaborations and its applications in life- and geo-science domains. The APE framework was successfully used to automate workflow composition in those domains, while the experiences from these applications provided valuable feedback which motivated APE development, including many of the existing features.

The APE framework demonstrates a favourable runtime performance. In addition, results from the survey of APE users show positive user experiences. Although the framework is still being actively developed and improved, the research presented answers the goals set by this dissertation. The provided SLTL^x-based formalism (1) supports direct integration with existing semantic domain annotations in life- and geo-sciences, (2) provides an intuitive problem specification format, (3) generates well-formatted and suitable candidate solutions, and finally, (4) it is distributed as a lightweight and portable library.

[4] `https://github.com/simonscheider/QuAnGIS/tree/master/WorkflowRepository`

Bibliography

[1] Abdala Asbun, A., Besseling, M. A., Balzano, S., Bleijswijk, J. D. L. van, Witte, H. J., Villanueva, L. & Engelmann, J. C., "Cascabel: A Scalable and Versatile Amplicon Sequence Data Analysis Pipeline Delivering Reproducible and Documented Results", *Frontiers in Genetics*, vol. 11, 2020, URL: https://www.frontiersin.org/article/10.3389/fgene.2020.489357 (visited on 06/03/2022).

[2] Afgan, E., Baker, D., Batut, B., denăBeek, M. van, Bouvier, D., čech, M., Chilton, J., Clements, D., Coraor, N., Grüning, B. A., Guerler, A., Hillman-Jackson, J., Hiltemann, S., Jalili, V., Rasche, H., Soranzo, N., Goecks, J., Taylor, J., Nekrutenko, A. & Blankenberg, D., "The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update", *Nucleic Acids Research*, vol. 46, no. W1, July 2018, W537–W544, DOI: 10.1093/nar/gky379, URL: https://academic.oup.com/nar/article/46/W1/W537/5001157 (visited on 09/27/2018).

[3] Al-Areqi, S., Lamprecht, A.-L. & Margaria, T., "Constraints-driven automatic geospatial service composition: workflows for the analysis of sea-level rise impacts", in: *Computational Science and Its Applications – ICCSA 2016*, ed. by Gervasi, O. et al., Cham: Springer International Publishing, 2016, pp. 134–150, ISBN: 978-3-319-42111-7.

[4] Albrecht, J., "Universal analytical GIS operations: A task-oriented systematization of data structure-independent GIS functionality", *Geographic information research: Transatlantic perspectives*, 1998, pp. 577–591.

[5] Alechina, N., Brázdil, T., De Giacomo, G., Felli, P., Logan, B. & Vardi, M. Y., "Unbounded orchestrations of transducers for manufacturing", in: *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2646–2653.

[6] Altintas, I., Crawl, D., Crosby, C. J. & Cornillon, P., "Scientific workflows for the geosciences: an emerging approach to building integrated data analysis systems", in: *Geoinformatics: Cyberinfrastructure for the Solid Earth Sciences*, ed. by Keller, G. R. & Baru, C., Cambridge University Press, 2011, pp. 237250, DOI: 10.1017/CBO9780511976308.016.

[7] Alur, R., Bodik, R., Juniwal, G., Martin, M. M. K., Raghothaman, M., Seshia, S. A., Singh, R., Solar-Lezama, A., Torlak, E. & Udupa, A., "Syntax-guided

synthesis", in: *2013 Formal Methods in Computer-Aided Design*, Oct. 2013, pp. 1–8.

[8] Amstutz, P., Crusoe, M. R., Tijanić, N., et al., "Common Workflow Language, v1.0", July 2016, DOI: 10.6084/m9.figshare.3115156.v2, URL: https://www.research.manchester.ac.uk/portal/en/publications/common-workflow-language-v10(741919f5-d0ab-4557-9763-b811e911423b)/publications.html (visited on 02/04/2020).

[9] Antoniou, G. & Harmelen, F. van, "Web Ontology Language: OWL", en, in: *Handbook on Ontologies*, ed. by Staab, S. & Studer, R., International Handbooks on Information Systems, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 67–92, (visited on 02/26/2019).

[10] Athanasis, N., Kalabokidis, K., Vaitis, M. & Soulakellis, N., "Towards a semantics-based approach in the development of geographic portals", *Computers & Geosciences*, vol. 35, no. 2, 2009, pp. 301–308.

[11] Atkinson, M., Gesing, S., Montagnat, J. & Taylor, I., "Scientific workflows: Past, present and future", *Future Generation Computer Systems*, vol. 75, Oct. 2017, pp. 216–227, DOI: 10.1016/j.future.2017.05.041, URL: http://www.sciencedirect.com/science/article/pii/S0167739X17311202 (visited on 08/02/2018).

[12] Atkinson, M., Gesing, S., Montagnat, J. & Taylor, I., "Scientific workflows: Past, present and future", *Future Generation Computer Systems*, vol. 75, 2017, pp. 216 –227, DOI: https://doi.org/10.1016/j.future.2017.05.041.

[13] Baker, C. J. O., Manir, M. S. A., Brenas, J. H., Zinszer, K. & Shaban-Nejad, A., *Applied Ontologies for Global Health Surveillance and Pandemic Intelligence*, en, Pages: 2020.10.17.20214460, Oct. 2020, DOI: 10.1101/2020.10.17.20214460, URL: https://www.medrxiv.org/content/10.1101/2020.10.17.20214460v1 (visited on 06/24/2022).

[14] Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D., *Manifesto for Agile Software Development*, 2001, URL: http://www.agilemanifesto.org/.

[15] Belardinelli, F., Lomuscio, A. & Patrizi, F., "Verification of agent-based artifact systems", *J. Artif. Intell. Res.*, vol. 51, 2014, pp. 333–376, DOI: 10.1613/jair.4424, URL: https://doi.org/10.1613/jair.4424.

[16] Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Thiel, K. & Wiswedel, B., "Knime-the konstanz information miner: version 2.0 and beyond", *AcM SIGKDD explorations Newsletter*, vol. 11, no. 1, 2009, pp. 26–31.

[17] Biere, A., Cimatti, A., Clarke, E. M., Strichman, O. & Zhu, Y., "Bounded model checking", *Advances in Computers*, vol. 58, 2003, pp. 117–148.

[18] *bio-tools/biotoolsSchema*, original-date: 2015-05-05T15:52:46Z, Dec. 2019, URL: https://github.com/bio-tools/biotoolsSchema (visited on 02/04/2020).

[19] Blair, D. C., "Information retrieval, 2nd ed. c.j. van rijsbergen. london: butterworths; 1979: 208 pp. price: $32.50", *Journal of the American Society for Information Science*, vol. 30, no. 6, 1979, pp. 374–375, DOI: 10.1002/asi.4630300621, eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/asi.4630300621, URL: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.4630300621.

[20] Bodik, R. & Jobstmann, B., "Algorithmic Program Synthesis: Introduction", *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 5, Oct. 2013, pp. 397–411, (visited on 09/12/2018).

[21] Brauner, J., "Formalizations for geooperators-geoprocessing in Spatial Data Infrastructures", PhD thesis, Technische Universität Dresden, 2015.

[22] Bylander, T., "The computational complexity of propositional STRIPS planning", *Artificial Intelligence*, vol. 69, no. 1-2, 1994, pp. 165–204.

[23] Calvanese, D., De Giacomo, G., Montali, M. & Patrizi, F., "Verification and synthesis in description logic based dynamic systems", in: *Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Proceedings*, ed. by Faber, W. & Lembo, D., vol. 7994, Lecture Notes in Computer Science, Springer, 2013, pp. 50–64, ISBN: 978-3-642-39665-6, DOI: 10.1007/978-3-642-39666-3, URL: https://doi.org/10.1007/978-3-642-39666-3.

[24] Calvanese, D., De Giacomo, G., Montali, M. & Patrizi, F., "First-order μ-calculus over generic transition systems and applications to the situation calculus", *Inf. Comput.*, vol. 259, no. 3, 2018, pp. 328–347, DOI: 10.1016/j.ic.2017.08.007, URL: https://doi.org/10.1016/j.ic.2017.08.007.

[25] Capella-Gutierrez, S., Iglesia, D. d. l., Haas, J., Lourenco, A., Fernández, J. M., Repchevsky, D., Dessimoz, C., Schwede, T., Notredame, C., Gelpi, J. L. & Valencia, A., "Lessons learned: recommendations for establishing critical periodic scientific benchmarking", *bioRxiv*, 2017, DOI: 10.1101/181677, eprint: https://www.biorxiv.org/content/early/2017/08/31/181677.full.pdf, URL: https://www.biorxiv.org/content/early/2017/08/31/181677.

[26] Chen, L., Shadbolt, N., Goble, C., et al., "Towards a Knowledge-Based Approach to Semantic Service Composition", in: *The SemanticWeb - ISWC 2003*, 2003, pp. 319334.

[27] Chrisman, N., "Exploring geographic information systems, 2nd edition", in: Wiley, 2002, chap. Transformations and operations, pp. 103–242.

[28] Clarke, E., Grumberg, O., Jha, S., Lu, Y. & Veith, H., "Counterexample-guided abstraction refinement", in: *Computer Aided Verification*, ed. by Emerson, E. A. & Sistla, A. P., Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 154–169, ISBN: 978-3-540-45047-4.

[29] Coyne, M. S. & Godley, B. J., "Satellite Tracking and Analysis Tool (STAT): an integrated system for archiving, analyzing and mapping animal tracking data", *Marine Ecology Progress Series*, vol. 301, Oct. 2005, pp. 1–7, (visited on 01/22/2019).

[30] De Giacomo, G., Lespérance, Y. & Patrizi, F., "Bounded situation calculus action theories", *Artif. Intell.*, vol. 237, 2016, pp. 172–203, DOI: 10.1016/j.

artint.2016.04.006, URL: https://doi.org/10.1016/j.artint.2016.04.006.

[31] De Giacomo, G. & Vardi, M. Y., "Synthesis for LTL and LDL on finite traces", in: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, ed. by Yang, Q. & Wooldridge, M. J., AAAI Press, 2015, pp. 1558–1564, URL: http://ijcai.org/proceedings/2015.

[32] De Giacomo, G., Vardi, M. Y., Felli, P., Alechina, N. & Logan, B., "Synthesis of orchestrations of transducers for manufacturing", in: *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[33] Di Tommaso, P., Chatzou, M., Floden, E. W. & others, "Nextflow enables reproducible computational workflows", *Nature Biotechnology*, vol. 35, Apr. 2017, pp. 316–319, DOI: 10.1038/nbt.3820.

[34] DiBernardo, M., Pottinger, R. & Wilkinson, M., "Semi-automatic web service composition for the life sciences using the biomoby semantic web framework", *Journal of Biomedical Informatics*, vol. 41, no. 5, 2008, Semantic Mashup of Biomedical Data, pp. 837–847, DOI: https://doi.org/10.1016/j.jbi.2008.02.005, URL: https://www.sciencedirect.com/science/article/pii/S1532046408000269.

[35] Dijkstra, E. W., "Program Inversion", in: *Program Construction, International Summer School*, London, UK, UK: Springer-Verlag, 1979, pp. 54–57, ISBN: 978-3-540-09251-3, (visited on 03/25/2019).

[36] Dingle, H., *Migration: the biology of life on the move*, Oxford University Press, USA, 2014.

[37] Dodge, K. L., Galuardi, B., Miller, T. J. & others, "Leatherback Turtle Movements, Dive Behavior, and Habitat Characteristics in Ecoregions of the Northwest Atlantic Ocean", *PLoS ONE*, vol. 9, no. 3, Mar. 2014, (visited on 01/22/2019).

[38] EEN, N., "MiniSat : A SAT solver with conflict-clause minimization", *Proc. SAT-05: 8th Int. Conf. on Theory and Applications of Satisfiability Testing*, 2005, pp. 502–518, (visited on 09/23/2018).

[39] Farnaghi, M. & Mansourian, A., "Disaster planning using automated composition of semantic ogc web services: a case study in sheltering", *Computers, Environment and Urban Systems*, vol. 41, 2013, pp. 204–218.

[40] Fitzner, D., Hoffmann, J. & Klien, E., "Functional description of geoprocessing services as conjunctive datalog queries", *GeoInformatica*, vol. 15, no. 1, 2011, pp. 191–221.

[41] Freitag, B., Steffen, B., Margaria, T. & Zukowski, U., "An Approach to Intelligent Software Library Management", in: *Proceedings of the 4th International Conference on Database Systems for Advanced Applications (DASFAA)*, World Scientific Press, 1995, pp. 7178, ISBN: 981-02-2220-3, URL: http://portal.acm.org/citation.cfm?id=646710.702986.

[42] Froleyks, N., Heule, M., Iser, M., Järvisalo, M. & Suda, M., "Sat competition 2020", *Artificial Intelligence*, vol. 301, 2021, p. 103572, DOI: https://doi.org/10.1016/j.artint.2021.103572, URL: https://www.sciencedirect.com/science/article/pii/S0004370221001235.

[43] Galton, A., "States, processes and events, and the ontology of causal relations", *Frontiers in Artificial Intelligence and Applications*, vol. 239, Jan. 2012, pp. 279–292, DOI: 10.3233/978-1-61499-084-0-279.

[44] Garijo, D., "AI Buzzwords Explained: Scientific Workflows", *AI Matters*, vol. 3, no. 1, May 2017, pp. 4–8, DOI: 10.1145/3054837.3054839, URL: http://doi.acm.org/10.1145/3054837.3054839 (visited on 10/24/2018).

[45] GDAL/OGR contributors, *GDAL/OGR Geospatial Data Abstraction software Library*, Open Source Geospatial Foundation, 2022, DOI: 10.5281/zenodo.5884351, URL: https://gdal.org.

[46] Gil, Y., Ratnakar, V., Kim, J. & others, "Wings: Intelligent Workflow-Based Design of Computational Experiments", *IEEE Intelligent Systems*, vol. 26, no. 1, Jan. 2011, pp. 62–72.

[47] Goecks, J., Nekrutenko, A., Taylor, J. & others, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences", *Genome Biology*, vol. 11, no. 8, Aug. 2010, R86, DOI: 10.1186/gb-2010-11-8-r86, URL: https://doi.org/10.1186/gb-2010-11-8-r86 (visited on 02/04/2020).

[48] Green, C., "Application of Theorem Proving to Problem Solving", in: *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, IJCAI'69, event-place: Washington, DC, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1969, pp. 219–239, (visited on 03/25/2019).

[49] Gulwani, S, "Programming by examples (and its applications in data wrangling)", in: Apr. 2016, pp. 137–158.

[50] Gulwani, S., "Dimensions in Program Synthesis", in: *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, PPDP '10, event-place: Hagenberg, Austria, New York, NY, USA: ACM, 2010, pp. 13–24, ISBN: 978-1-4503-0132-9, (visited on 03/21/2019).

[51] Gulwani, S., "Automating String Processing in Spreadsheets Using Input-output Examples", in: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, event-place: Austin, Texas, USA, New York, NY, USA: ACM, 2011, pp. 317–330, ISBN: 978-1-4503-0490-0, (visited on 03/25/2019).

[52] Gulwani, S., Jha, S., Tiwari, A. & Venkatesan, R., "Synthesis of Loop-free Programs", in: *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, event-place: San Jose, California, USA, New York, NY, USA: ACM, 2011, pp. 62–73, ISBN: 978-1-4503-0663-8, (visited on 03/21/2019).

[53] Gulwani, S., Polozov, O. & Singh, R., *Program Synthesis*, English, vol. 4, Foundations and Trends in Programming Languages 1-2, now, July 2017, URL: https://www.nowpublishers.com/article/Details/PGL-010 (visited on 08/02/2018).

[54] Guptill, S. C. & Morrison, J. L., *Elements of spatial data quality*, Elsevier, 2013.

[55] Haverkort, K., Kasalica, V., Eelman, S., Janse, A., Rademaker, M., Tjoeng, M., Dam, I. van, Wal, J. van der, Nieuwenhuizen, S. van, Wolters, R., Wouts, S. & Lamprecht, A.-L., *sanctuuary/APE-Web: APE Web 1.3.3*, version v1.3.3, Nov. 2021, DOI: 10.5281/zenodo.5638953, URL: https://doi.org/10.5281/zenodo.5638953.

[56] Henriksen, J. G., Jensen, J., Jørgensen, M., Klarlund, N., Paige, R., Rauhe, T. & Sandholm, A., "Mona: monadic second-order logic in practice", in: *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 1995, pp. 89–110.

[57] Hofer, B., Mäs, S., Brauner, J. & Bernard, L., "Towards a knowledge base to support geoprocessing workflow development", *International Journal of Geographical Information Science*, vol. 31, no. 4, 2017, pp. 694–716.

[58] Hofer, B., Papadakis, E. & Mäs, S., "Coupling knowledge with gis operations: the benefits of extended operation descriptions", *ISPRS International Journal of Geo-Information*, vol. 6, no. 2, 2017, p. 40.

[59] Hopcroft, J. E., Motwani, R. & Ullman, J. D., "Introduction to automata theory, languages, and computation, 2nd edition", *ACM SIGACT News*, vol. 32, no. 1, Mar. 2001, pp. 60–65, DOI: 10.1145/568438.568455, URL: https://doi.org/10.1145/568438.568455 (visited on 07/23/2020).

[60] Iannopollo, A., *ianno/pyco: SCP2018*, version SCP2018, Nov. 2017, DOI: 10.5281/zenodo.1066685, URL: https://doi.org/10.5281/zenodo.1066685.

[61] Iannopollo, A., Tripakis, S. & Sangiovanni-Vincentelli, A., "Constrained Synthesis from Component Libraries", en, in: *Formal Aspects of Component Software*, ed. by Kouchnarenko, O. & Khosravi, R., Lecture Notes in Computer Science, Cham: Springer International Publishing, 2017, pp. 92–110, ISBN: 978-3-319-57666-4, DOI: 10.1007/978-3-319-57666-4_7.

[62] Ilkay Altintas Chad Berkley, E. J. et al., "Kepler: An Extensible System for Design and Execution of Scientific Workflows", in: *Proceedings of SSDBM 2004*, IEEE Computer Society, 2004, pp. 2123, DOI: 10.1.1.5.9905.

[63] Ison, J., Ioan, H., Rydza, E., Chmura, P., Rapacki, K., Gaignard, A., Schwämmle, V., Helden, J. van, Kalaš, M. & Ménager, H., "Biotoolsschema: a formalized schema for bioinformatics software description", *GigaScience*, vol. 10, Jan. 2021, DOI: 10.1093/gigascience/giaa157.

[64] Ison, J., Kalaš, M., Jonassen, I., et al., "EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats", *Bioinformatics*, 2013.

[65] Ison, J., Ménager, H., Brancotte, B., Jaaniso, E., Salumets, A., Raček, T., Lamprecht, A.-L., Palmblad, M., Kalaš, M., Chmura, P., Hancock, J. M., Schwämmle, V. & Ienasescu, H.-I., "Community curation of bioinformatics software and data resources", *Briefings in Bioinformatics*, Oct. 2019, bbz075, DOI: 10.1093/bib/bbz075, eprint: https://academic.oup.com/bib/article-pdf/doi/10.1093/bib/bbz075/30157114/bbz075.pdf, URL: https://doi.org/10.1093/bib/bbz075.

[66] Ison, J., Rapacki, K., Ménager, H. & others, "Tools and data services registry: a community effort to document bioinformatics resources", *Nucleic Acids Research*, vol. 44, no. D1, Jan. 2016, pp. D38–47.

[67] Ison, J. C., Ménager, H., Brancotte, B., Jaaniso, E., Salumets, A., Racek, T., Lamprecht, A., Palmblad, M., Kalas, M., Chmura, P., Hancock, J. M., Schwämmle, V. & Ienasescu, H., "Community curation of bioinformatics software and data resources", *Briefings Bioinform.*, vol. 21, no. 5, 2020, pp. 1697–1705, DOI: 10.1093/bib/bbz075, URL: https://doi.org/10.1093/bib/bbz075.

[68] Jesus, J. de, Walker, P., Grant, M. & Groom, S., "Wps orchestration using the taverna workbench: the escience approach", *Computers & Geosciences*, vol. 47, 2012, pp. 75–86.

[69] Jha, S., Gulwani, S., Seshia, S. A. & Tiwari, A., "Oracle-guided Component-based Program Synthesis", in: *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, event-place: Cape Town, South Africa, New York, NY, USA: ACM, 2010, pp. 215–224, ISBN: 978-1-60558-719-6, DOI: 10.1145/1806799.1806833, URL: http://doi.acm.org/10.1145/1806799.1806833 (visited on 03/22/2019).

[70] Johnston, K., Ver Hoef, J. M., Krivoruchko, K. & Lucas, N., *Using ArcGIS geostatistical analyst*, vol. 380, Esri Redlands, 2001.

[71] Jones, C. B. & Purves, R. S., "Geographical information retrieval", *International Journal of Geographical Information Science*, vol. 22, no. 3, 2008, pp. 219–228.

[72] Kasalica, V., Alechina, N., Lamprecht, A.-L. & Logan, B., "Instance-Aware Synthesis of Workflows Specified in Temporal Logic", *Journal of Artificial Intelligence Research (JAIR)*, 2023, Submitted and under review.

[73] Kasalica, V. & Lamprecht, A.-L., "Workflow Discovery Through Semantic Constraints: A Geovisualization Case Study", in: *Computational Science and Its Applications – ICCSA 2019*, Cham: Springer International Publishing, 2019, pp. 473–488, ISBN: 978-3-030-24302-9.

[74] Kasalica, V. & Lamprecht, A.-L., "APE: A Command-Line Tool and API for Automated Workflow Composition", in: *Computational Science – ICCS 2020*, ed. by Krzhizhanovskaya, V. V., Závodszky, G., Lees, M. H., Dongarra, J. J., Sloot, P. M. A., Brissos, S. & Teixeira, J., Cham: Springer International Publishing, 2020, pp. 464–476, ISBN: 978-3-030-50436-6.

[75] Kasalica, V. & Lamprecht, A.-L., "Workflow Discovery with Semantic Constraints: The SAT-Based Implementation of APE", *Electronic Communications of the EASST*, vol. 78, May 2020, DOI: 10.14279/tuj.eceasst.78.1092, URL: https://journal.ub.tu-berlin.de/eceasst/article/view/1092 (visited on 05/17/2020).

[76] Kasalica, V., Schwämmle, V., Palmblad, M., Ison, J. & Lamprecht, A.-L., "APE in the Wild: Automated Exploration of Proteomics Workflows in the bio.tools Registry", *Journal of Proteome Research*, vol. 20, no. 4, 2021, PMID: 33720735, pp. 2157–2165, DOI: 10.1021/acs.jproteome.0c00983, eprint: https://doi.org/10.1021/acs.jproteome.0c00983, URL: https://doi.org/10.1021/acs.jproteome.0c00983.

[77] Kautz, H. & Selman, B., "Planning as satisfiability", in: *Proceedings of the 10th European conference on Artificial intelligence*, ECAI '92, Vienna, Austria:

John Wiley & Sons, Inc., Aug. 1992, pp. 359–363, ISBN: 978-0-471-93608-4, (visited on 02/01/2020).

[78] Kautz, H. & Selman, B., "Pushing the envelope: planning, propositional logic, and stochastic search", in: *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 2*, AAAI'96, Portland, Oregon: AAAI Press, Aug. 1996, pp. 1194–1201, ISBN: 978-0-262-51091-2, (visited on 02/01/2020).

[79] Kazemi Beydokhti, M., Duckham, M., Griffin, A. & Kasalica, V., "Geo-Event Question Answering Systems: A Preliminary Research Study", Sept. 2021, DOI: 10.25436/E2KW2T, URL: https://escholarship.org/uc/item/9cs309kd (visited on 11/01/2021).

[80] Kleyheeg, E., Dijk, J. G. B. van, Tsopoglou-Gkina, D. & others, "Movement patterns of a keystone waterbird species are highly predictable from landscape configuration", *Movement Ecology*, vol. 5, no. 1, Feb. 2017, p. 2, (visited on 02/05/2019).

[81] Kona, S., Bansal, A., Blake, M. & Gupta, G., "Generalized Semantics-Based Service Composition", in: *ICWS 2008*, IEEE Computer Society, Sept. 2008, pp. 219–227.

[82] Köster, J. & Rahmann, S., "Snakemakea scalable bioinformatics workflow engine", *Bioinformatics*, vol. 28, no. 19, Oct. 2012, pp. 2520–2522, DOI: 10.1093/bioinformatics/bts480, URL: https://academic.oup.com/bioinformatics/article/28/19/2520/290322 (visited on 02/04/2020).

[83] Kranstauber, B., Cameron, A., Weinzerl, R., Fountain, T., Tilak, S., Wikelski, M. & Kays, R., "The Movebank data model for animal tracking", *Environmental Modelling & Software*, vol. 26, no. 6, June 2011, pp. 834–835, (visited on 02/15/2019).

[84] Kruiger, J. F., Kasalica, V., Meerlo, R., Lamprecht, A.-L., Nyamsuren, E. & Scheider, S., "Loose programming of GIS workflows with geo-analytical concepts", *Transactions in GIS*, vol. 25, no. 1, 2021, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.12692, pp. 424–449, DOI: 10.1111/tgis.12692, URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/tgis.12692 (visited on 09/02/2021).

[85] Kuhn, W., "Core concepts of spatial information for transdisciplinary research", *International Journal of Geographical Information Science*, vol. 26, no. 12, 2012, pp. 2267–2276.

[86] Kuhn, W. & Ballatore, A., "Designing a language for spatial computing", in: *AGILE 2015*, Springer, 2015, pp. 309–326.

[87] Kumar, A., Rasche, H., Grüning, B. & Backofen, R., "Tool recommender system in Galaxy using deep learning", *GigaScience*, vol. 10, no. 1, Jan. 2021, giaa152, DOI: 10.1093/gigascience/giaa152, eprint: https://academic.oup.com/gigascience/article-pdf/10/1/giaa152/35531829/giaa152_reviewer_4_report_original_submission.pdf, URL: https://doi.org/10.1093/gigascience/giaa152.

[88] Kuncak, V., Mayer, M., Piskac, R., Suter, P., Kuncak, V., Mayer, M., Piskac, R. & Suter, P., "Complete functional synthesis", *ACM SIGPLAN Notices*, vol. 45, no. 6, June 2010, pp. 316–329, (visited on 03/22/2019).

[89] Lamprecht, A.-L., *User-Level Workflow Design - A Bioinformatics Perspective*, vol. 8311, Lecture Notes in Computer Science, Springer, 2013, pp. 1–202, ISBN: 978-3-642-45388-5, 978-3-642-45389-2, DOI: 10.1007/978-3-642-45389-2.

[90] Lamprecht, A.-L., Margaria, T. & Steffen, B., "Bio-jETI: a framework for semantics-based service composition", *BMC Bioinformatics*, vol. 10 Suppl 10, 2009, S8.

[91] Lamprecht, A.-L., Naujokat, S., Margaria, T. & Steffen, B., "Synthesis-Based Loose Programming", in: *Proc. of the 7th Int. Conf. on the Quality of Information and Communications Technology (QUATIC 2010), Porto, Portugal*, IEEE, Sept. 2010, pp. 262–267.

[92] Lamprecht, A.-L., Palmblad, M., Ison, J., Schwämmle, V., Manir, M. S. A., Altintas, I., Baker, C. J. O., Amor, A. B. H., Capella-Gutierrez, S., Charonyktakis, P., Crusoe, M. R., Gil, Y., Goble, C., Griffin, T. J., Groth, P., Ienasescu, H., Jagtap, P., Kalaš, M., Kasalica, V., Khanteymoori, A., Kuhn, T., Mei, H., Ménager, H., Möller, S., Richardson, R. A., Robert, V., Soiland-Reyes, S., Stevens, R., Szaniszlo, S., Verberne, S., Verhoeven, A. & Wolstencroft, K., *Perspectives on automated composition of workflows in the life sciences*, en, tech. rep. 10:897, Type: article, F1000Research, Sept. 2021, DOI: 10.12688/f1000research.54159.1, URL: https://f1000research.com/articles/10-897 (visited on 01/19/2022).

[93] Lau, T. A., Domingos, P. & Weld, D. S., "Version Space Algebra and Its Application to Programming by Demonstration", in: *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 527–534, ISBN: 978-1-55860-707-1, (visited on 03/22/2019).

[94] Lemmens, R., Wytzisk, A., By, R., Granell, C., Gould, M. & Oosterom, P. van, "Integrating semantic and syntactic descriptions to chain geographic services", *IEEE Internet Computing*, vol. 10, no. 5, 2006, pp. 42–52.

[95] Liu, J., Pacitti, E., Valduriez, P., et al., "A Survey of Data-Intensive Scientific Workflow Management", *Journal of Grid Computing*, vol. 13, no. 4, 2015, pp. 457–493, DOI: 10.1007/s10723-015-9329-8.

[96] Ludäscher, B., Lin, K., Bowers, S., Jaeger-Frank, E., Brodaric, B. & Baru, C., "Managing scientific data: from data integration to scientific workflows", *Geoinformatics: Data to knowledge*, vol. 397, 2006, p. 109.

[97] Lustig, Y. & Vardi, M. Y., "Synthesis from Component Libraries", en, in: *Foundations of Software Science and Computational Structures*, ed. by Alfaro, L. de, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 395–409, ISBN: 978-3-642-00596-1.

[98] Lutz, M., "Ontology-based descriptions for semantic discovery and composition of geoprocessing services", *Geoinformatica*, vol. 11, no. 1, 2007, pp. 1–36.

[99] Malik, S. & Zhang, L., "Boolean satisfiability from theoretical hardness to practical success", *Commun. ACM*, vol. 52, no. 8, Aug. 2009, pp. 7682, DOI: 10.1145/1536616.1536637, URL: https://doi.org/10.1145/1536616.1536637.

[100] Mandelin, D., Xu, L., Bodík, R. & Kimelman, D., "Jungloid Mining: Helping to Navigate the API Jungle", in: *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, event-place: Chicago, IL, USA, New York, NY, USA: ACM, 2005, pp. 48–61, ISBN: 978-1-59593-056-9, (visited on 03/22/2019).

[101] Manna, Z. & Waldinger, R., "Knowledge and reasoning in program synthesis", *Artificial Intelligence*, vol. 6, no. 2, June 1975, pp. 175–208, (visited on 03/25/2019).

[102] Manna, Z. & Waldinger, R. J., "Toward Automatic Program Synthesis", *Commun. ACM*, vol. 14, no. 3, Mar. 1971, pp. 151–165, (visited on 03/25/2019).

[103] Margaria, T. & Steffen, B., "LTL-Guided Planning: Revisiting Automatic Tool Composition in ETI", in: *Proc. of the 31st Annual IEEE / NASA Software Engineering Workshop (SEW 2007), Columbia, MD, USA*, IEEE Computer Society, 2007, pp. 214–226, ISBN: 0-7695-2862-7, URL: http://portal.acm.org/citation.cfm?id=1338445.1338873&coll=GUIDE&dl=GUIDE.

[104] Margaria, T. & Steffen, B., "Service-Orientation: Conquering Complexity with XMDD", English, in: *Conquering Complexity*, ed. by Hinchey, M. & Coyle, L., Springer London, 2012, pp. 217–236, ISBN: 978-1-4471-2296-8, DOI: 10.1007/978-1-4471-2297-5_10, URL: http://dx.doi.org/10.1007/978-1-4471-2297-5_10.

[105] Martin, D., Paolucci, M., McIlraith, S., et al., "Bringing Semantics to Web Services: The OWL-S Approach", in: *Semantic Web Services and Web Process Composition*, vol. 3387, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2005, pp. 26–42, DOI: 10.1007/b105145.

[106] MO, D. H. & WITTEN, I. H., "Learning text editing tasks from examples: a procedural approach", *Behaviour & Information Technology*, vol. 11, no. 1, Jan. 1992, pp. 32–45, (visited on 03/22/2019).

[107] Müller, M., "Hierarchical profiling of geoprocessing services", *Computers & Geosciences*, vol. 82, 2015, pp. 68–77.

[108] Naujokat, S., Lamprecht, A.-L. & Steffen, B., "Loose Programming with PROPHETS", in: *Proc. of FASE 2012, Estonia*, vol. 7212, LNCS, 2012, pp. 94–98, DOI: 10.1007/978-3-642-28872-2_7.

[109] Naujokat, S., Lybecait, M., Kopetzki, D. & Steffen, B., "CINCO: A Simplicity-Driven Approach to Full Generation of Domain-Specific Graphical Modeling Tools", *Software Tools for Technology Transfer*, 2017, DOI: 10.1007/s10009-017-0453-6.

[110] Naujokat, S., Neubauer, J., Margaria, T. & Steffen, B., "Meta-Level Reuse for Mastering Domain Specialization", in: *Proc. of the 7th Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Part II (ISoLA 2016)*, vol. 9953, LNCS, Springer, 2016, pp. 218–237, DOI: 10.1007/978-3-319-47169-3_16.

[111] Palmblad, M., Lamprecht, A.-L., Ison, J. & Schwämmle, V., "Automated workflow composition in mass spectrometry-based proteomics", 2018.

[112] Phothilimthana, P. M., Thakur, A., Bodik, R. & others, "Scaling up Superoptimization", *ACM SIGOPS Operating Systems Review*, vol. 50, no. 2, June 2016, pp. 297–310, (visited on 03/21/2019).

[113] Pnueli, A. & Rosner, R., "On the Synthesis of a Reactive Module", in: *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, New York, NY, USA: ACM, 1989, pp. 179–190, ISBN: 978-0-89791-294-5, (visited on 09/12/2018).

[114] Pnueli, A. & Rosner, R., "Distributed Reactive Systems Are Hard to Synthesize", in: *FOCS*, 1990.

[115] Riazanov, A., Klein, A., Shaban-Nejad, A., Rose, G. W., Forster, A. J., Buckeridge, D. L. & Baker, C. J., "Semantic querying of relational data for clinical intelligence: a semantic web services-based approach", *Journal of Biomedical Semantics*, vol. 4, no. 1, Mar. 2013, p. 9, DOI: 10.1186/2041-1480-4-9.

[116] Ríos, J., Karlsson, J. & Trelles, O., "Magallanes: a web services discovery and automatic workflow composition tool", *BMC Bioinformatics*, vol. 10, no. 1, Oct. 2009, p. 334, DOI: 10.1186/1471-2105-10-334, URL: https://doi.org/10.1186/1471-2105-10-334 (visited on 06/26/2022).

[117] Scheider, S. & Ballatore, A., "Semantic typing of linked geoprocessing workflows", *International Journal of Digital Earth*, vol. 11, no. 1, 2018, pp. 113–138.

[118] Scheider, S., Gräler, B., Pebesma, E. & Stasch, C., "Modeling spatiotemporal information generation", *International Journal of Geographical Information Science*, vol. 30, no. 10, 2016, pp. 1980–2008.

[119] Scheider, S. & Jong, T. de, "A conceptual model for automating spatial network analysis", *Transactions in GIS*, vol. 26, no. 1, 2022, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.12855, pp. 421–458, DOI: 10.1111/tgis.12855, URL: http://onlinelibrary.wiley.com/doi/abs/10.1111/tgis.12855 (visited on 06/03/2022).

[120] Scheider, S., Meerlo, R., Kasalica, V. & Lamprecht, A.-L., "Ontology of Core Concept Data Types for Answering Geo-Analytical Questions", *Journal of Spatial Information Science*, vol. 2020, no. 20, 2020, pp. 167–201, DOI: 10.5311/JOSIS.2020.20.555, URL: https://digitalcommons.library.umaine.edu/josis/vol2020/iss20/2 (visited on 02/04/2022).

[121] Scheider, S., Ostermann, F. O. & Adams, B., "Why good data analysts need to be critical synthesists. determining the role of semantics in data analysis", *Future generation computer systems*, vol. 72, 2017, pp. 11–22.

[122] Schilling, S., "Prototype for natural language queries for GIS analyses", MA thesis, Ruprecht-Karls-University Heidelberg, Germany, 2021, p. 100.

[123] Schrijvers, T., Tack, G., Wuille, P., Samulowitz, H. & Stuckey, P. J., "An Introduction to Search Combinators", in: *International Symposium on Logic-Based Program Synthesis and Transformation*, Springer, 2012, pp. 2–16.

[124] Shapiro, E. Y., *Algorithmic Program Debugging*, en, Aug. 2004, DOI: 10.7551/mitpress/1192.001.0001, URL: https://direct.mit.edu/books/book/4799/Algorithmic-Program-Debugging (visited on 06/30/2022).

[125] Shaw, D. E., Swartout, W. R. & Green, C. C., *Inferring LISP Programs from Examples*, en, tech. rep. CUCS-001-75, Department of Computer Science, Columbia University, 1975.

[126] Smith, D. C., *Pygmalion: A Creative Programming Environment*, en, Google-Books-ID: mihHAAAAIAAJ, Computer Science Department, Stanford University, 1975.

[127] Solar-Lezama, A., "Program Synthesis by Sketching", PhD Thesis, Berkeley, CA, USA: University of California at Berkeley, 2008.

[128] Srivastava, S., Gulwani, S. & Foster, J. S., "From program verification to program synthesis", in: *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2010)*, Madrid, Spain: ACM, 2010, pp. 313–326.

[129] Steffen, B., Margaria, T. & Braun, V., "The Electronic Tool Integration platform: concepts and design", *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1-2, 1997, pp. 9–30, DOI: 10.1007/s100090050003.

[130] Steffen, B., Margaria, T. & Freitag, B., *Module Configuration by Minimal Model Construction*, tech. rep., Fakultät für Mathematik und Informatik, Universität Passau, 1993.

[131] Steffen, B., Margaria, T., Nagel, R., Jörges, S. & Kubczak, C., "Model-Driven Development with the jABC", in: *Hardware and Software, Verification and Testing*, ed. by Bin, E., Ziv, A. & Ur, S., vol. 4383, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2007, pp. 92–108, ISBN: 978-3-540-70888-9, DOI: 10.1007/978-3-540-70889-6_7.

[132] Summers, P. D., "A Methodology for LISP Program Construction from Examples", en, in: *Readings in Artificial Intelligence and Software Engineering*, ed. by Rich, C. & Waters, R. C., Morgan Kaufmann, Jan. 1986, pp. 309–316, ISBN: 978-0-934613-12-5.

[133] Sun, Z., Yue, P., Lu, X., Zhai, X. & Hu, L., "A task ontology driven approach for live geoprocessing in a service-oriented environment", *Transactions in GIS*, vol. 16, no. 6, 2012, pp. 867–884.

[134] Taylor, I. J., Deelman, E., Gannon, D. B. & Shields, M., *Workflows for E-Science: Scientific Workflows for Grids*, Springer Publishing Company, Incorporated, 2014, ISBN: 1849966192.

[135] Twidale, M. & Hansen, P., "Agile research", *First Monday*, vol. 24, no. 1, Jan. 2019, DOI: 10.5210/fm.v24i1.9424, URL: https://journals.uic.edu/ojs/index.php/fm/article/view/9424.

[136] Uschold, M. & Jasper, R., "A framework for understanding and classifying ontology applications", in: *Proceedings 12th Int. Workshop on Knowledge Acquisition, Modelling, and Management KAW*, vol. 99, 1999, pp. 16–21.

[137] Valmari, A., "The State Explosion Problem", in: *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets*, London, UK: Springer-Verlag, 1998, pp. 429–528, ISBN: 3-540-65306-6.

[138] Vandervalk, B. P., McCarthy, E. L. & Wilkinson, M. D., "SHARE: A Semantic Web Query Engine for Bioinformatics", en, in: *The Semantic Web*, ed. by Gómez-Pérez, A., Yu, Y. & Ding, Y., Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2009, pp. 367–369, ISBN: 978-3-642-10871-6, DOI: 10.1007/978-3-642-10871-6_27.

[139] Visser, U., Stuckenschmidt, H., Schuster, G. & Vogele, T., "Ontologies for geographic information processing", *Computers & Geosciences*, vol. 28, 2002, pp. 103–117, DOI: 10.1016/S0098-3004(01)00019-X.

[140] Vivian, J., Rao, A. A., Nothaft, F. A., Ketchum, C., Armstrong, J., Novak, A., Pfeil, J., Narkizian, J., Deran, A. D., Musselman-Brown, A., Schmidt, H., Amstutz, P., Craft, B., Goldman, M., Rosenbloom, K., Cline, M., O'Connor, B., Hanna, M., Birger, C., Kent, W. J., Patterson, D. A., Joseph, A. D., Zhu, J., Zaranek, S., Getz, G., Haussler, D. & Paten, B., "Toil enables reproducible, open source, big biomedical data analyses", *Nature Biotechnology*, vol. 35, no. 4, Apr. 2017, pp. 314–316, DOI: 10.1038/nbt.3772, URL: http://www.nature.com/articles/nbt.3772 (visited on 04/15/2020).

[141] Wessel, P. & Smith, W. H. F., "Free software helps map and display data", *EOS Trans. Amer. Geophys. U.*, vol. 72, no. 41, 1991.

[142] Wessel, P., Smith, W. H., Scharroo, R. & others, "Generic mapping tools: improved version released", *EOS Trans. Amer. Geophys. U.*, vol. 94, no. 45, 2013, pp. 409–410.

[143] Wiegand, N. & García, C., "A task-based ontology approach to automate geospatial data retrieval", *Transactions in GIS*, vol. 11, no. 3, 2007, pp. 355–376.

[144] Wilkinson, M. D., Vandervalk, B. & McCarthy, L., "The Semantic Automated Discovery and Integration (SADI) Web service Design-Pattern, API and Reference Implementation", *Journal of Biomedical Semantics*, vol. 2, no. 1, Oct. 2011, p. 8, DOI: 10.1186/2041-1480-2-8, URL: https://doi.org/10.1186/2041-1480-2-8 (visited on 02/18/2019).

[145] Withers, D., Kawas, E., McCarthy, L., Vandervalk, B. & Wilkinson, M., "Semantically-guided workflow construction in taverna: the sadi and biomoby plug-ins", in: *Leveraging Applications of Formal Methods, Verification, and Validation*, ed. by Margaria, T. & Steffen, B., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 301–312, ISBN: 978-3-642-16558-0.

[146] Witt, M., Åkesson, S., Broderick, A., Coyne, M., Ellick, J, Formia, A, Hays, G., Luschi, P., Stroud, S., Godley, B., et al., "Assessing accuracy and utility of satellite-tracking data using argos-linked fastloc-gps", *Animal Behaviour*, vol. 80, no. 3, 2010, p. 571.

[147] Wolstencroft, K., Haines, R., Fellows, D., et al., "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud", *Nucleic Acids Research*, vol. 41, no. W1, 2013, W557–W561.

[148] *Workflow Description Language (WDL)*, original-date: 2012-08-01T03:12:48Z, Apr. 2020, URL: https://github.com/openwdl/wdl (visited on 04/15/2020).

[149] Yue, P., Baumann, P., Bugbee, K. & Jiang, L., "Towards intelligent giservices", *Earth Science Informatics*, vol. 8, no. 3, 2015, pp. 463–481.

[150] Yue, P., Di, L., Yang, W., Yu, G. & Zhao, P., "Semantics-based automatic composition of geospatial web service chains", *Computers & Geosciences*, vol. 33, no. 5, 2007, pp. 649–665.

[151] Zhuang, C., Xie, Z., Ma, K., Guo, M. & Wu, L., “A task-oriented knowledge base for geospatial problem-solving”, *ISPRS International Journal of Geo-Information*, vol. 7, no. 11, 2018, p. 423.

Summary

The last two decades brought an explosion of computational tools and processes in many scientific domains (e.g., life-, social- and geo-science). Scientific workflows, i.e., computational pipelines, accompanied by workflow management systems, were soon adopted as a de-facto standard among non-computer scientists for orchestrating such computational processes. The goal of this dissertation is to provide a framework that would automate the orchestration of such computational pipelines in practice. We refer to such problems as scientific workflow synthesis problems.

This dissertation extends an existing temporal logic-based workflow synthesis approach. The original approach was not able to keep track of data instances within the workflow and to describe data flow dependencies, i.e., to describe relations between tools and data. Such limitations can substantially hinder the applicability of a synthesis approach. This dissertation introduces the extended, SLTL^x-based synthesis to overcome the known limitations of the original formalism. The new approach uses transducers and temporal goals, which keep track of the data objects in the synthesised workflow. The proposed SLTL^x-based synthesis includes a *bounded* and a *dynamic* variant, which are shown in Chapter 3 to be NP-complete and PSPACE-complete, respectively.

Chapter 4 introduces a transformation algorithm that translates the bounded SLTL^x-based synthesis problem into propositional logic. The transformation is implemented as part of the APE (Automated Pipeline Explorer) framework, presented in Chapter 5. It relies on highly efficient SAT solving techniques, using an off-the-shelf SAT solver to synthesise a solution for the given propositional encoding. The framework provides an API (application programming interface), a CLI (command line interface), and a web-based GUI (graphical user interface).

The development of APE was accompanied by four concrete application scenarios as case studies for automated workflow composition. The studies were conducted in collaboration with domain experts and presented in Chapter 6. Each of the case studies is used to assess and illustrate specific features of the SLTL^x-based synthesis approach. (1) A case study on cartographic map generation demonstrates the ability to distinguish data objects as a key feature of the framework. It illustrates the process of annotating a new domain, and presents the iterative workflow synthesis approach, where the user tries to narrow down the desired specification of the problem in a few intuitive steps. (2) A case study on geo-analytical question answering as part of the QuAnGIS project shows the benefits of using data flow dependencies

to describe a synthesis problem. In addition, the study shows good synthesis results when used on a well-annotated domain. (3) A proteomics case study demonstrates the usability of APE as an "off-the-shelf" synthesiser, providing direct integration with existing semantic domain annotations. In addition, a manual evaluation of the synthesised results shows promising results even on large real-life domains, such as the EDAM ontology and the complete bio.tools registry. (4) A geo-event question answering study demonstrates the usability of APE within a larger question answering system.

The experiences from these applications, in particular the feedback from the involved domain experts, influenced the design decisions during the development of the APE framework. They motivated the development of existing features, such as the possibility of using multiple disjoint *semantic dimensions* to model data, and providing workflow solutions in the CWL (Common Workflow Language) format. The framework is still being actively developed and improved.

The APE framework demonstrates a favourable runtime performance, in all of the presented case studies. In addition, the runtime was positively assessed as part of a survey of APE users, presented in Chapter 7. The survey results further emphasise the importance of a scientific workflow synthesis library for automated question answering. In addition, they show positive user experiences with the framework documentation, formats of the synthesised results, and the provided interfaces.

This dissertation answers the goals it sets to solve. It provides a formal framework, accompanied by a lightweight library, which can solve real-life scientific workflow synthesis problems. Finally, the development of the library motivated an upcoming collaborative project in the life sciences domain. The aim of the project is to develop a platform which would automatically compose (using APE) and benchmark workflows in computational proteomics.

Samenvatting

De afgelopen twee decennia hebben een explosie teweeggebracht van computationele tools en processen in allerlei wetenschappelijke domeinen (zoals levens, sociale en geowetenschappen). Wetenschappelijke workflows, d.w.z. computationele pijplijnen, tezamen met workflow managementsystemen, werden al snel aangenomen als een de-facto standaard onder niet-computerwetenschappers voor het orkestreren van zulke computationele processen. Het doel van dit proefschrift is om een framework te bieden dat de orkestratie van zulke computationele pijplijnen in de praktijk automatiseert. Dergelijke problemen noemen we wetenschappelijke workflow-syntheseproblemen.

Dit proefschrift breidt een bestaande, op temporele logica gebaseerde, workflow-synthesebenadering uit. De oorspronkelijke aanpak was niet in staat om data-instanties binnen de workflow bij te houden en om de afhankelijkheden van datastromen te beschrijven, oftewel, om relaties tussen tools en data te beschrijven. Deze beperkingen kunnen de toepasbaarheid van een synthesebenadering aanzienlijk belemmeren. Dit proefschrift introduceert de uitgebreide op SLTL^x gebaseerde synthese, om de bekende beperkingen van het oorspronkelijke formalisme te boven te komen. De nieuwe aanpak maakt gebruik van transducers en temporele doelen die de data-objecten in de gesynthetiseerde workflow bijhouden. De voorgestelde op SLTL^x gebaseerde synthese omvat een *begrensde* en een *dynamische* variant, die in Hoofdstuk 3 aangetoond worden respectievelijk NP-compleet en PSPACE-compleet te zijn.

Hoofdstuk 4 introduceert een transformatie-algoritme dat het begrensde op SLTL^x gebaseerde syntheseprobleem vertaalt naar de propositielogica. De transformatie is geïmplementeerd als onderdeel van het APE (Automated Pipeline Explorer) framework, gepresenteerd in Hoofdstuk 5. Het is gebaseerd op zeer efficiënte SAT-solving technieken, waarbij gebruik wordt gemaakt van een kant-en-klare SAT-solver om een oplossing voor de propositionele codering te synthetiseren. Het framework biedt een API (application programming interface), een CLI (command line interface) en een webgebaseerde GUI (graphical user interface).

De ontwikkeling van APE ging gepaard met vier concrete toepassingsscenario's als casestudies voor geautomatiseerde workflowsamenstelling. De casestudies zijn uitgevoerd in samenwerking met domeinexperts en gepresenteerd in Hoofdstuk 6. Elk van de casestudies wordt gebruikt om specifieke kenmerken van de op SLTL^x gebaseerde synthesebenadering te illustreren en beoordelen. (1) Een casestudy over

het genereren van cartografische kaarten toont het vermogen aan om dataobjecten te onderscheiden als zijnde een belangrijk kenmerk van het framework. Het illustreert het proces van het annoteren van een nieuw domein en presenteert de iteratieve workflow-synthesebenadering, waarbij de gebruiker de gewenste specificatie van het probleem in een aantal intuïtieve stappen probeert te verfijnen. (2) Een casestudy aangaande het beantwoorden van geo-analytische vragen, als onderdeel van het QuAnGIS-project, toont de voordelen van het gebruik van datastroom afhankelijkheden in het beschrijven van een syntheseprobleem. Bovendien laat de studie goede syntheseresultaten zien bij het gebruik op een goed geannoteerd domein. (3) Een casestudy in proteomics demonstreert de bruikbaarheid van APE als een kant-en-klare synthesizer, die directe integratie biedt met bestaande semantische domeinannotaties. Bovendien laat een handmatige evaluatie van de gesynthetiseerde resultaten veelbelovende resultaten zien, zelfs op uitgebreide real-life domeinen, zoals de EDAM-ontologie en het volledige bio.tools-register. (4) Een casestudy in het beantwoorden van vragen over geo-events toont de bruikbaarheid van APE aan binnen een groter systeem aangaande het beantwoorden van vragen.

De ervaringen van deze toepassingen, in het bijzonder de feedback van de betrokken domeinexperts, hebben invloed gehad op de ontwerpbeslissingen tijdens de ontwikkeling van het APE-framework. Zij hebben de ontwikkeling van aanwezige features gemotiveerd, zoals de mogelijkheid om meerdere onsamenhangende *semantische dimensies* te gebruiken om data te modelleren, en het bieden van workflowoplossingen in het CWL-format (Common Workflow Language). Het framework wordt nog steeds actief ontwikkeld en verbeterd.

Het APE-framework laat gunstige runtime-prestaties zien in alle gepresenteerde casestudies. Bovendien werd de runtime positief beoordeeld als onderdeel van een onderzoek onder APE-gebruikers, gepresenteerd in Hoofdstuk 7. Ook benadrukken de onderzoeksresultaten het belang van een wetenschappelijke workflow-synthesebibliotheek voor het geautomatiseerd beantwoorden van vragen. Daarnaast laten de resultaten positieve gebruikerservaringen zien met de documentatie van het framework, formats van de gesynthetiseerde resultaten en de geleverde interfaces.

Dit proefschrift beantwoordt de doelen die het tracht op te lossen. Het biedt een formeel framework, samen met een lichtgewicht bibliotheek, dat real-life wetenschappelijke workflow-syntheseproblemen kan oplossen. Ten slotte heeft de ontwikkeling van de bibliotheek geleid tot een toekomstig samenwerkingsproject in het domein van de levenswetenschappen. Het doel van het project is om een platform te ontwikkelen om automatisch workflows in computationele proteomics samen te stellen (met behulp van APE) en te benchmarken.

Curriculum Vitae

Vedran Kasalica
Born on April 2nd, 1992 in Kotor, Montenegro.

Education

2018 – 2022	*Universiteit Utrecht*, Netherlands PhD in Computer Science
2015 – 2017	*TU Dresden*, Germany *Free University of Bozen-Bolzano*, Italy *Universidade Nova de Lisboa*, Portugal Master in Computational Logic
2011 – 2015	*University of Montenegro*, Montenegro Bachelor in Computer Science

Work Experience

2022 –	Research Software Engineer *eScience Center*, Netherlands
2017 – 2019	Co-founder, Software Architect *See Side Montenegro*, Montenegro
2014 – 2015	Software Engineer *Logate d.o.o.*, Montenegro

Acknowledgments

This dissertation describes my scientific contributions in the past four years, but it does not capture all my attempts, iterations and failures, all the discussions, advice and coffee chats with my colleagues, and all the trips, games nights and drinks with friends. These interactions allowed me to finalise this dissertation and successfully close this chapter of my life. This section is dedicated to all of you.

First and foremost, I would like to thank my supervisor, Anna-Lena, who was an instrumental part of my PhD journey. Thank you for giving me the opportunity and enough freedom to explore this field according to my interests, and for having patience for our countless discussions, both on and off the research topic. Apart from sharing your expertise in the field, you thought me to always try to look at things from more than one perspective. Thank you, it helped me in both, my professional and my personal life.

I want to extend my gratitude to Gabriele Keller, for accepting to be my promotor and for her crucial input that shaped this dissertation. I also want to thank my collaborators, Simon Scheider, Enkhbold Nyamsuren, Han Kruiger, Veit Schwämmle, Magnus Palmblad, Natasha Alechina, Brian Logan and Mohammad Kazemi Beydokhti, for helping me find and solve exciting research problems, and motivating me to push it further each day. In addition, I would like to thank Maurin Voshol, Koen Haverkort and the SP team that worked on APE Web, for your contributions to the APE framework. Finally, I want to thank Christian, Gianni and Elena for taking the time to read and give me feedback on this manuscript, and Mariëlle for translating the summary into Dutch, I know that it was not easy, and I really appreciate it.

When I joined UU the ST group welcomed me with open arms. Thank you Gabi, Johan, Wishnu, Jeroen, Jurriaan, Raja, Nico, Sergey, Anna-Lena, Victor, Alejandro, Joao, Trevor, Matthijs, Ivo, Iris, Isaac, Samira, Saba, Jacco, Tom, David, Fernando (I hope I didn't miss anyone) for many passionate discussions during the Monday lunch meetings. After such a meeting, Victor and Joao introduced me to the Wednesday drinks and UU-ING (International Neighbours Group of UU). This ended up being the place where I met most of my friends here in the Netherlands. I will be eternally grateful for that to UU-ING. Thank you Nazmiye and Agnes for agreeing to take over the torch with me and host the Jan Primus drinks! We could have been even better than our predecessors, but unfortunately, a pandemic got in our way :)

To my paranymphs. Isaac, we started our PhDs practically at the same time and it

was always a relief knowing that there is someone going through similar phases of the PhD with me. Your numerous advice and suggestions, regarding Elena's visa, our apartment, Nasho and Ljova, to name a few, drastically shaped my life in Utrecht, and I will always be grateful for that! Victor, you introduced me not only to the ING, but to the beauties of craft beers, wines and many *grachts* of Utrecht. Thank you for all the kayaking, gaming, and wine/beer drinking sessions. For all the chats, discussions and advice about my research as well as my future/current jobs. Finally, thank you for helping me to find my way in this confusing world of countless vaseline and powder options, my feet are eternally grateful. Thank you both for being there with me on this journey!

I would like to thank Filip for showing me the beauties of the Netherlands six years ago and convincing me to consider moving here after defending my MSc thesis. Doing my PhD research was challenging, however, I enjoyed the whole process. To be honest, coping with the pandemic felt like the biggest challenge in the process. That is why I would like to thank all the friends that shared that experience with me, which brought us even closer. Chris, Gianni, Miloše, Najo and Ela (*u stomaku*), thank you for all those visits to Verdansk, I really needed those trips, while the Dutch airports were closed. Isaac and Jose, you made the first waves of the pandemic bearable. The game and movie nights were the social interaction we desperately needed in those times, and the avondklok loophole made those gatherings even sweeter! Thank you for introducing us to Museo, Maxima and eventually to the extended family, Max, Billie and Jeroen! Victor and Nadine, thank you for the countless Pandemic and Gloomhaven nights! It all started with you bringing that one game. If I had known, I would have bought it sooner! I hope we get that basement setup one day. Lisa, Kristina thank you for being part of all the dinners and weekend trips! Alvaro, I am glad that we stayed in touch after you moved and now we got to extend the group with Masha and Azul as well. Sofia, Anto, Marina, I like to think that the bitter taste of that last GoT season is what pushed us into all those fantastic nights out, Eurovisions, and birthday parties. Marielle, Dave thank you for playing with us all sorts of games, cycling to the beach and selflessly sharing your fantastic stash of food and wine. Aco, I'm really glad you decided to move here. It feels like there were no breaks between our time in Podgorica and now in the Netherlands.

Working at the university has its perks, and 42 days of vacation was definitely one of them. Naturally, that meant I had plenty of time to go on holiday. Gianni, Chris, our yearly trips were definitely among the highlights! Great food and music, and the unavoidable *Harmonika* would always ensure a well-deserved relaxing holiday and something to always look forward to! Trips to Montenegro where I would see my friends and family were naturally also a yearly tradition. *Braća (i sestre koje su realno braća) bi uvjek izdvojila vrijeme da se gledamo koliko god možemo dok bih bio dolje. Jeste da se nismo gledali koliko bi zeljeli, ali ste vazda doekali brata na gallu. Ljubi brat braću ludu!*

None of this would be possible without my flatmates, Nasho and Ljova. They kept me inspired and entertained, and crucially, they taught me that there is always something more important than my problems - their problems. Finally, I would like to thank my third flatmate, Elena. She was the one who found the advertisement

for this PhD programme and helped me apply for it. So, Elena, it's hard to imagine where I would be without you right now. Walking Camino was an eye-opening experience, but doing it with you made it extraordinary. We tested and pushed our limits together, and in the process got closer than ever before. I cherish equally the first steps in Porto as the last steps in Santiago, but unlike that 10 days walk, our Camino has just begun. *Buen Camino*!

The last paragraph is *posvećen mojim roditeljima i sestri. Znam da se ne gledamo koliko bi željeli, ali mislim na vas češće nego što mislite. Hvala vam na svoj podršci, ne samo tokom doktorata, već tokom svih mojih godina odrastanja. Mislim da je suvišno govoriti o vašoj ulozi. Naučili ste me da nikada ne treba osuđivati druge, da treba vjerovati u sebe i ostvariti svoje snove! Iva, znam da su naše poznanstvo poelo sa puno svađa i suza, ali kako je vrijeme odmicalo toga je bilo sve manje, a mi smo postajali sve bliži. Volim vas.*

DOI *https://doi.org/10.33540/160*
Cover design based on illustrations by *Greenwingstudio*.