Universiteit Utrecht

Master Thesis

From logic-based to algebra-based specification of geo-analytical workflows

Author: Stan van Meerendonk
Student Number: 4284763
Program: Master Artificial Intelligence

Supervisors
Dr. A.L. Lamprecht
Dr. S.W.B. Prasetya

Co-Supervisors
Vedran Kasalica MSc
Dr. Simon Scheider

September 10, 2020

Abstract

This thesis explores the use of a domain-specific algebra as an input for a SAT-based workflow synthesis system. With this algebra, I try to make the gap smaller from logic-based input to natural-language-based input for the use of workflow synthesis systems in the Geoscience domain. The thesis extends the Automatic Pipeline Explorer (APE) to use the Geospatial Analytical Transformation Algebra (GATA) as an extra input. The goal is to let a user specify their intent using a GATA expression and APE will find a correct workflow that recreates that input. This is possible because components of workflows also can be represented by GATA. A workflow is correct if It represents the input GATA expression.

To accomplish this, GATA-adapted APE transforms the additional input and domain information into CNF constraints so that APE can use this information. I use the structure of the expressions to define constraints to use in APE. I use the clear order within a GATA expression to define sets of constraints.

This approach can only approximate the correct workflows. To ensure only correct workflows appear in the result, a post-process step is added. It reconstructs the GATA expression of the workflow and compares it to the workflow specification. If the specification expression cannot be recreated from a workflow, the workflow is discarded. Only the correct workflows remain.

I tested GATA-adapted APE on four ArcGIS examples and compared it to the APE approach. The new approach resulted in a smaller amount but more domain-specific workflows. This makes it easier for a user to find a workflow if they provide a GATA expression. However, the time needed to generate the constraints and to post-process the results increases drastically for more extended workflows.

The code related to this thesis can be found on the following repositories.

- `https://github.com/sstuber/APE_Extension`
- `https://github.com/sstuber/APE`

Contents

1 Introduction

Science is becoming increasingly more involved with complex calculations. This forces scientists to use computational programs to let a computer execute the calculations. A common use case in the geo-analytical domain is combining geographical data to create more complex maps. One example is calculating the livability index for elderly people [19]. This can be calculated by combining geographical data regarding noise levels (Figure 1), population and facilities of the neighbourhoods, like the distance to a green area (Figure 2). Calculating the livability index by hand is not a trivial task.

Usually, the calculation for a use case consists of a sequence of already available components, such as computational components, or program functions. Such sequence of components is called a workflow (Figure 3). The scientific community has worked on solutions to assist scientists in creating these workflows. Workflow management systems assist the user in developing workflows. Examples of such systems include Apache Taverna [25], Kepler [3] and ArcGIS ModelBuilder [2]. These systems assist the user by creating a clear overview of the workflow and assist the user in composing them. However, a user needs to have knowledge of the available components of their domain to work effectively and efficiently.

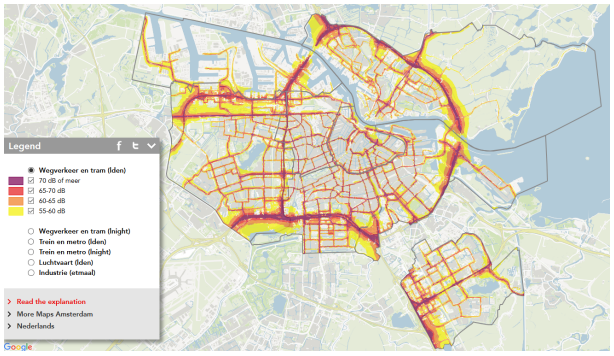

Figure 1: A visual representation of the noise levels in the city center of Amsterdam

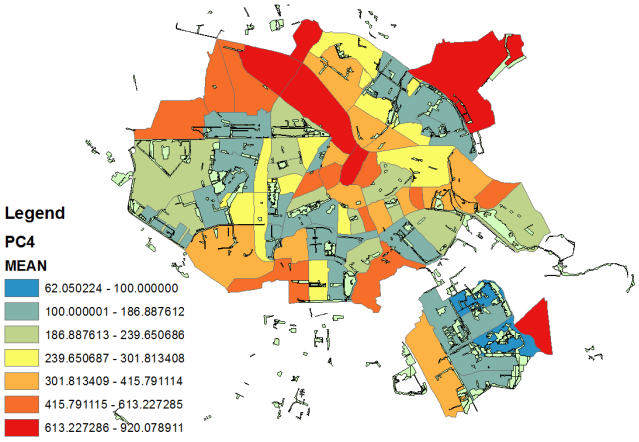

Figure 2: Livability map showing average distance to green areas in Amsterdam within PC4 areas

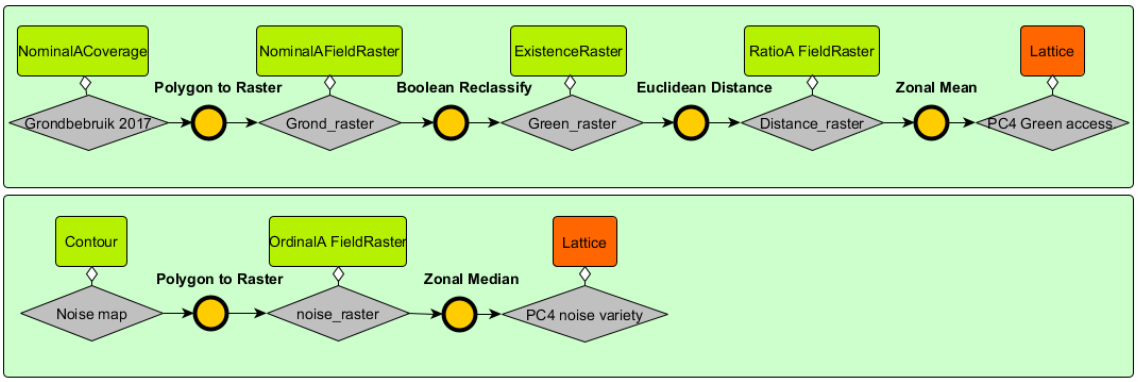

Figure 3: Two example workflows regarding calculating the livability index. The top workflow calculates the average distance to green areas, which the result can be found in Figure 2. The bottom workflow calculates the noise variety, which uses Figure 1 as input. This figure is borrowed from [19]

A new user might have an idea what they want to compute. To support the users with workflow composition, these workflow management systems can be integrated with workflow synthesis systems. A system like this takes user created constraints and generates a possible sequence of components, a workflow. The synthesis process handles the selection of components and combining them into workflows. This way, a user does not need full knowledge of all relevant tools.

Common input methods for workflow synthesis systems are natural language templates or visual models. DECLARE [16] is an example of visual modeling and PROPHETS [14] is an example of natural language templates. In these systems, the given templates and models represent some form of predetermined temporal formula reasoning over concrete or abstract tools. Examples of this are "Use tool1" or "If tool1 is used tool2 also has to be used". Modeling systems have similar constraints represented by visual models. These temporal constraints are then used to find workflows that satisfy them.

These current approaches of defining user intent abstract away from the direct implementations. This makes it easier for the user to explain their goal to the system. However, the abstraction is based on function and type descriptions. These descriptions use domain specific data types and operations. In some cases, it would be easier for a user to describe their intent in a more semantically describing way. What if we can abstract further from the the data types and operations?

The goal of this thesis is to explore the usage of such abstraction. I used the following research questions:

1. How can input semantically describe the goal?
2. How can this input be used in a workflow synthesis system?
3. How can it relate such input to concrete tools?
4. How can we translate the semantic input and domain knowledge into constraints for a SAT-based system?

The thesis explores the use of Geo-Analytical Transformation Algebra (GATA) [18] which is an algebra currently being developed. It reasons over abstract core concepts [12] using relational algebra, as is often used in Geoscience. The language describes relations over abstract concepts and not their direct implementation. An expression in the languages describes the computation and its final product semantically. This allows the user to write their intent in terms of the required calculations without the need to specify the domain-specific tool abstractions. The same user intent specification can be used for different domains like ArcGIS or python.

This thesis is split into six sections. Section 2 discusses the work this thesis is built upon. Section 3 introduces the terminology that is used in the rest of the thesis. Section 4 discusses the approach used to implement the research questions. Section 5 reviews the results of the mentioned approach and compares it to the approach it is based upon. Finally, Section 6 concludes the thesis.

2 Related Work

Workflow synthesis is a subdomain of program synthesis. With program synthesis, the user defines the required input and output and defines additional constraints on the behaviour of the program. An example is Sketch [21], which also reduces the problem to a set of fomulas that can be solved by a SAT solver. The user creates a program with holes and creates constraints on the values. The synthesis algorithm then translates this into a formula that is solvable by a SAT solver. This constraint-based approach is common in program synthesis [10]. However there are multiple approaches to the problems, user intent, search space and search technique, of program synthesis [9].

The workflow synthesis system [13], based on the minimal modal algorithm [22] is a similar approach. They loosely specify the constraints on their domain using Semantic Linear Temporal Logic (SLTL) and a type system. The SLTL constraints are then used to search the synthesis domain using the minimal modal algorithm to find valid workflows. This algorithm has been implemented in the PROPHETS system [14].

The Automatic Pipeline Explorer (APE) works similar to this synthesis algorithm. It takes the semantic domain knowledge and user intent as workflow specification and generates type correct workflows. APE translates the problem into CNF and uses MiniSAT [7].

Both PROPHETS and APE use natural language templates as the input for the user intent. These templates represent a formula in SLTL [22]. SLTL is an extension of Linear Temporal Logic (LTL) [8]. This extension allows to the use of abstract types and modules in the specification. SLTL allows express constraints over a type correct sequence of tools therefore it is the user intent representation for APE. This form of user intent reasons direct over tools or abstractions of tools.

Multiple case studies have shown the potential of workflow synthesis. For instance, a case study [11] has shown APE is effective in the domain of Geoscience. A different case study [15] has shown effectiveness with PROPHETS within the domain of bioinformatics. Both papers show the usage of such workflow synthesis systems. They make the usability of the respective domains easier by lowering the prerequisite knowledge of all available tools.

There have been cases where natural language was used as input for program synthesis [6] [17] and where workflow synthesis was automatised completely [26]. However, there has not been a case where an algebra or program has been used as input for workflow synthesis. GATA expresses the required calculations based on the core concepts semantically. The operations used in GATA are based in relational algebra [5]. This too has not been used as a user intent specification.

3 Prerequisite Knowledge

This section introduces the required terminology. The first section introduces APE, the Automatic Pipeline Explorer. The thesis expands on APE. The second section introduces the GATA language. The third section introduces a formal notation of graphs. This notation is used to reason over GATA. The final section discusses a set notation for the summation of formulas. This is used to define sets of constraints.

3.1 APE

APE is a tool that generates workflows from a workflow specification. Such specification contains the workflow input, workflow output and the constraints. A specification determines what workflows APE returns. To reason over tools, APE needs domain knowledge. This domain knowledge includes taxonomies and tool annotations. An overview of the input and output of APE can be found in Figure 4.

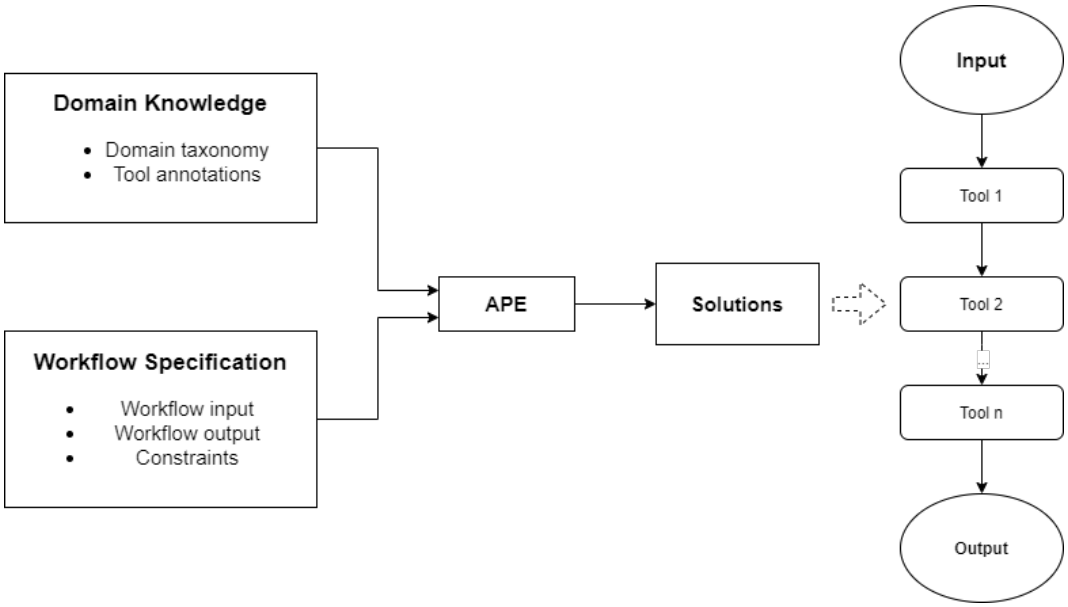

Figure 4: This figure shows the general input-output flow of APE. APE takes the domain knowledge and the workflow specification and generates workflows that adhere to the specification. A single workflow is shown to display the structure.

3.1.1 Domain Knowledge

The domain knowledge consists of several taxonomies and tools annotated with types from those taxonomies. A taxonomy describes the hierarchical structure of abstract things. In this case, it describes the abstract hierarchy of the tool and the data types. Figure 5 shows the taxonomies for the tools, the types and the formats of the ImageMagick example as found on the APE repository [1]. Usually the tool taxonomy only shows the abstract tool descriptions, but in this figure the concrete tools are included. You can see that each taxonomy has a strict hierarchy. For example, *PNG* is a subclass of *Lossless*. *Lossless* is a subclass of *ImageFormat* which is a subclass of the most general class *Format*.

The data type taxonomies help combine the input and outputs of the tools. If a tool requires a *Lossless* input, anything lower in the hierarchy will also fit, like *PNG* or *BMP*. If a tool has *Lossless* as output, it can be used as anything higher in the hierarchy, like *ImageFormat*. This is also the case with the tool taxonomy. If a user requests a *Distort* tool, any tool lower in the hierarchy will fit, like *Sharpen* or *Blur*.

The tool annotations make use of these taxonomies. Where the taxonomies describe the data and tool hierarchy, the tool annotations relate the concrete tools to abstract tools. Each tool is annotated with a taxonomy operation, input data types and output data types. This is the information APE uses to find tools for a workflow. An example of a single annotation can be found in Table 1

Label	Data
Id	cut
Taxonomy operation	Layers
Input	Type: Image or Type: Filter
Output	Type: Image and Format: PNG

Table 1: An example of an APE tool annotation

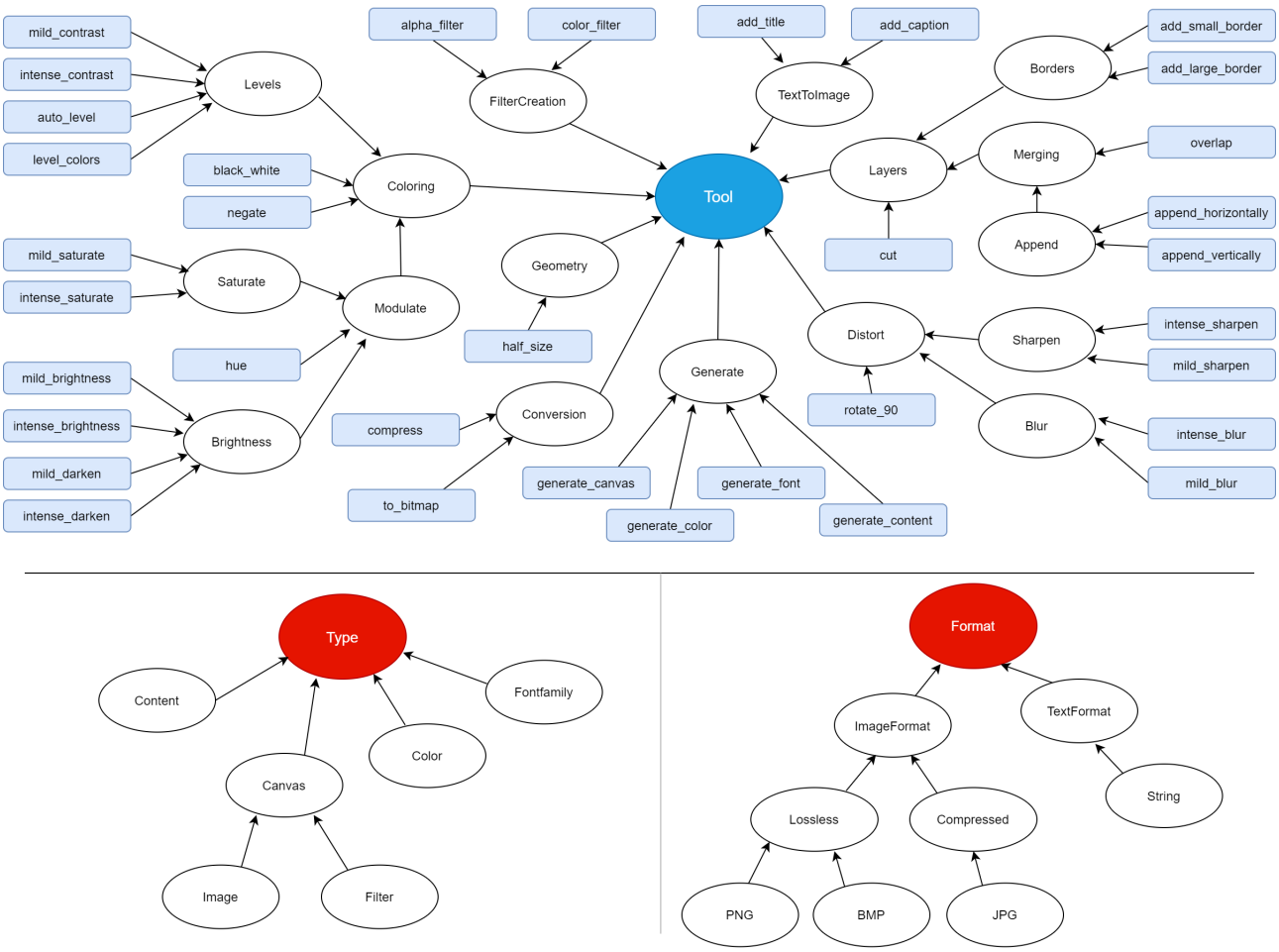

Figure 5: This figure shows the taxonomies for the Image Magick example. At the top, you see the tool taxonomy with "Tool" as the root. The white ovals represent the abstractions of tools and the blue squares represent the concrete tools. The bottom two pictures represent the datatype taxonomies.

3.1.2 Workflow Specification

The workflow specification consists of input data types, output data types and constraints. The input and output types need to be from the data types taxonomy. The constraints are additional information the user can give APE. These constraints are represented using SLTL.

SLTL is interpreted as alternating sequences of types and tools (also called modules). SLTL extends the LTL formalism to the BNF that can be found in Expression 1. These alternating sequences can be described as $t_0,\ m_1,\ t_1,\ m_2,\ t_2$. Here m_1 is the first tool in the sequence. These sequences can be simplified by leaving out the types as the types are implicit. The previous example will then look like m_1, m_2. These sequences of states are called paths. SLTL can be interpreted as potentially infinite paths.

$$\Phi := \ true \mid t_c \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \ < m_c > \Phi \mid \mathbf{G}\Phi \mid \mathbf{F}\Phi \mid \Phi\mathbf{U}\Phi \mid \tag{1}$$

- $true$ is satisfied by all paths regardless of their elements.
- t_c is satisfied by all paths where the first element satisfies the given t_c. Within APE, all subtypes of t_c are also allowed.

- $\neg$ and $\wedge$ are interpreted as their meaning in propositional logic. The $\vee$ operator can be expressed as $\neg(\neg\Phi \wedge \neg\Phi)$ but is included to be complete.
- $< m_c > \Phi$ is satisfied by paths were the second element (and thus the first tool element) satisfies m_c. all sub paths must satisfy ϕ.
- $\mathbf{G}\Phi$ is satisfied by all paths where Φ holds globally. There is no state where Φ does not hold
- $\Phi\mathbf{U}\Psi$ is satisfied by all paths where Φ holds until Ψ holds in the next state. This Strong until operator guarantees that Ψ at some point will be satisfied.
- $\mathbf{F}\Phi$ is satisfied by all paths where Φ becomes eventually true. It guarantees that Φ happens at least one time. This operator can be expressed in terms of $\mathbf{U}$ like $true\ \mathbf{U}\ \Phi$.

The constraints in APE are described in SLTL. However, users themselves do not directly use the SLTL to create their constraints. APE provides natural language templates that represent the given constraints. This allows users from all domains to use APE. A relevant selection of the given constraints can be found below.

Natural language template	SLTL constraint
If Tool_1 is used, Tool_2 has to be used subsequently	$\mathbf{G}(\neg < \text{Tool\_1} > true \vee \mathbf{XF} < \text{Tool\_2} > true)$
If Tool_1 is used, Tool_2 cannot be used subsequently	$\mathbf{G}(\neg < \text{Tool\_1} > true \vee \mathbf{XG}\neg < \text{Tool\_2} > true)$
If Tool_1 is used, Tool_2 must has to be used next in the sequence	$\mathbf{G}(\neg < \text{Tool\_1} > true \vee \mathbf{X} < \text{Tool\_2} > true)$
Use tool Tool_1 in the solution	$\mathbf{F} < \text{Tool\_1} > true$
Do not use tool Tool_1 in the solution	$\mathbf{G}\neg < \text{Tool\_1} > true$
Use Tool_1 as last tool in the solution	$\mathbf{F} < \text{Tool\_1} > \wedge \mathbf{G}(\neg < \text{Tool\_1} > true \ \vee \neg\mathbf{XX}\ true)$

Table 2: The natural language templates and their SLTL constraints as used by APE

To use these constraints in a SAT-based algorithm like APE, these constraints need to be translated into a propositional formula. In this translation [4] the notion $[\![f]\!]_n^i$ refers to the interpretation of the formula f in the i-th state of a sequence of length n. Each proposition consists of two components, the name of the proposition p and index i of the module state in which the proposition holds m_i. The resulting formula consists of propositions like $p(m_i)$ to note which state a proposition should hold. For example $reify(m_3)$ represents that $reify$ should hold in module state 3. A composed expression like $p(m_i)$ can only be true or false as a whole. The translation can be found below.

$$
\begin{aligned}
[\![\langle p \rangle true]\!]_n^i &:= p(m_{i+1}) \\
[\![\neg f]\!]_n^i &:= \neg [\![f]\!]_n^i \\
[\![f \vee g]\!]_n^i &:= [\![f]\!]_n^i \vee [\![g]\!]_n^i \\
[\![f \wedge g]\!]_n^i &:= [\![f]\!]_n^i \wedge [\![g]\!]_n^i \\
[\![\mathbf{G} f]\!]_n^i &:= [\![f]\!]_n^i \wedge [\![\mathbf{G} f]\!]_n^{i+1} \\
[\![\mathbf{F} f]\!]_n^i &:= [\![f]\!]_n^i \vee [\![\mathbf{F} f]\!]_n^{i+1} \\
[\![\mathbf{X} f]\!]_n^i &:= [\![f]\!]_n^{i+1} \\
[\![f]\!]_n^n &:= 0
\end{aligned}
$$

The interpretation for a single type is left out as it is not necessary for the rest of the thesis. A simple example of such translation can be found below.

$$[\![\mathbf{F} < example\_tool > true]\!]_2^0 = example\_tool(m_1) \vee example\_tool(m_2)$$

These translated propositional formulas can be transformed into CNF. This is the required input for the SAT solver that APE uses.

3.2 GATA

Geo-analytical question-answering (QuAnGIS) [20] tries to find a solution for a domain-specific natural-language-based question. In this area, GATA serves as a language that expresses the answer of a question. GATA describes a calculation with the core concepts [12] as a basis. It is defined with a simplified relational algebra as a formal basis. It reasons over relations with zero to two keys. Any of these relations are some representation of a core concept. GATA consists of a set of functions that represent operations over these relations. However, for this synthesis, the structure of the language is more important. If a reconstructed GATA representation of a workflow is the same GATA representation of the workflow specification, they represent the same operations. To determine whether two GATA expressions are the same, we need to look at the structure of the expressions. The GATA grammar is described by the BNF shown in Formula 2. Here, S is the start symbol of the grammar and id represents a string of the set of GATA functions when used as a function or a name that represents some form of input value.

$$S ::= id(S) \mid id(S,\ S) \mid id \tag{2}$$

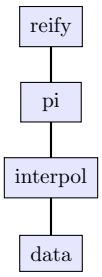

Figure 6: The parse tree representation of Expression 3, the simple example

$$reify\ (pi\ (interpol\ (input))) \tag{3}$$

The language consists of function identifiers with one or more parameters and identifiers that represent some form of input. An example of this would be Equation 3 with a graphical representation in Figure 6. This expression is the easiest type of expressions you can build with the GATA grammar. In this easy example, $reify$, pi and $interpol$ are function identifiers that represent a data transformation.

A real example is given by Formula 4. This is the GATA representation of one of the cases I use to evaluate the approach. A figure of the parse tree can be found in Figure 7. This example also shows full usage of the earlier mentioned BNF.

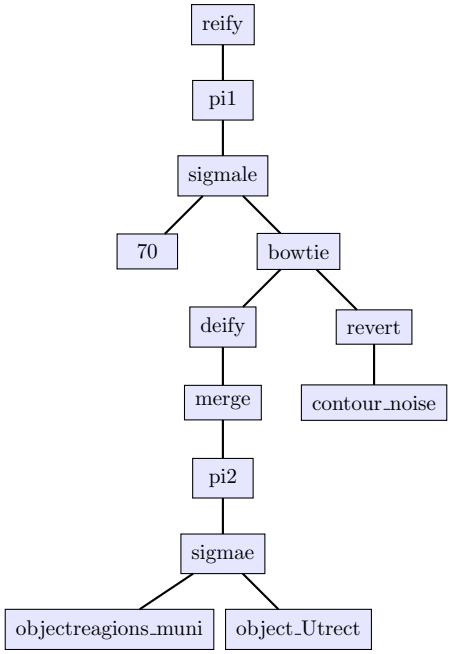

Figure 7: The parse tree of the representative GATA expression from Formula 4

$$reify(pi1(sigmale(bowtie(revert(contour\_noise),\\ deify(merge(pi2(sigmae(objectregions\_muni, object\_Utrecht))))), 70))) \quad (4)$$

Each function has a specific type signature. For instance, the function *reify* has as type signature $reify : L \rightarrow SV$ where the input type L refers to a set of locations and the output type SV refers to a single region. An expression is type correct when for every function, the input types that a function is given from the output of another function or as available data are the same as the required types for that function. Thus the output type of pi is the same as the input type of *reify* as can be seen in Expression 3.

For this thesis I assume the all GATA expressions that are used as specification are type correct as expressions can be verified on type correctness by a parser. The goal is to find a workflow that has a GATA representation that is the same as target expression. When the GATA expression in the workflow specification is type correct and a workflow has a GATA expression representation that matches it, the workflow will also be type correct with regards to GATA.

3.3 Graph Formalism

Using the BNF (2) to parse GATA will create a parse tree. The earlier mentioned easy (Figure 6) and real (Figure 7) show examples of such parse trees. The nodes in this tree are functions from the GATA language. Except for the leaves at the bottom, these represent data. The parse tree is build from top to bottom while

taking the most outer layer from the expression. However, the data flows from bottom to top. Each function can only be applied when all its subtrees below are applied.

To be able to use this tree as a basis for defining constraints I use the formal definition of a graph [23]. A graph G consists of a pair (V, E) where V is the set of vertices or nodes and E is the set of edges. Each edge is a pair of nodes like (n_1, n_2). When reasoning over GATA, the order of the functions matters. To reason over a GATA tree, the edges go from root to the leaves in the direction the expression is parsed. This means that the root of the tree does not appear as the second node in an edge and the leaves do not appear as the first node in an edge. Using this notation the input graph I would be represented like Equations 5

$$\begin{aligned} I &= (V, E) \\ V &= \{reify, pi, interpol, data\} \\ E &= \{(reify, pi), (pi, interpol), (interpol, data)\} \end{aligned} \tag{5}$$

Finally, nodes are considered unique. This holds even when different nodes represent the same function and thus have the same name. Functions can occur multiple times in the same expression. The same name appearing multiple times is not allowed in sets. However, this could be circumvented by giving all nodes an id number and making the nodes a tuple of id and function name. This would increase the complexity of the given definitions and thus will be omitted. Thus, when nodes are directly compared, for example ($n1$ equals $n2$), only the names of the nodes are evaluated instead of the whole nodes themselves.

3.4 Definitions

The first definition I introduce is the general set that contains all functions in the current domain, this set is called F. The tool specifications from the workflow specifications in APE also have a GATA annotation. The set of these tool annotations is called A. This set A consists of pairs (t, g) where t is the identifier of the tool as in APE and g is the tree representation of a GATA expression. An example of this set can be found in Expression 6. Each line represents one annotation. First you see the name of the tool, then the set of the nodes in the related parse tree and finally the set of edges in the related parse tree.

$$\begin{aligned} A_{example} = \{&(\text{reifyTool}, && (\{reify\}, && \{\})), \\ &(\text{combinedTool}, && (\{pi, interpol\}, && \{(pi, interpol)\}))\} \end{aligned} \tag{6}$$

The next definition is the parse tree of the GATA expression in the workflow specification. This is referred to as i. The nodes of this tree are referred to as V_i and the edges are referred to as E_i.

The constraints are defined using set comprehensions. This definition is adapted from [24]. These comprehensions define sets of SLTL formulas that represent certain constraints. The BNF represent the set comprehensions can be found at Definition 7

$$\begin{aligned} &\textit{SLTL} && \Phi ::= true \mid t_c \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \ < m_c > \Phi \mid \mathbf{G}\Phi \mid \mathbf{F}\Phi \mid \Phi\mathbf{U}\Phi \mid \\ &\textit{Basic elements} && E ::= \Phi \mid String \mid Int \mid Boolean \mid Func \mid T \mid x \\ &\textit{Tuples} && T ::= (All\ \langle, All\rangle^*) \\ &\textit{Sets of formulas} && S ::= \{\Phi \mid Fl\langle, Fl\rangle^*\} \mid \emptyset \mid S \cup S \mid S \cap S \mid v \\ &Formulas && Fl ::= \neg Fl \mid Fl \wedge Fl \mid Fl \vee Fl \mid E == E \mid E \in S \mid E \in Func \\ &\textit{All types} && All ::= E \mid S \\ &\textit{Function} && Func ::= Ident(\ param\ \langle, param\rangle^*) = All \end{aligned} \tag{7}$$

The important thing to note here is the tuples of this BNF can also contain sets. This is necessary to reason over the tool annotations. Another introduction is general use functions. Some constraints are recursively created thus introduced the need for recursive functions. The Φ in basic elements represents the SLTL structures defined earlier. Furthermore, I use a function *count* that returns the amount of elements in the set. These set comprehensions are used to reason over the GATA tree of the input and over the set of GATA annotations to create sets of SLTL formulas.

4 Approach

This section addresses how I approached the problem. First, I explain the problem in more detail. A simple example is used to give an idea what the result of the process should be. Then I show how I address the problem using the simple example. This should give an idea how it is built up. The following two sections expand on the simple example. I expand the simple example to an example that uses functions with multiple parameters, then I discuss how tools that have no GATA representation are included. The final section discusses a post-processing step that ensures only workflows match with the input are returned.

4.1 Problem Introduction

To use GATA expressions as constraints for APE, I evaluated how GATA relates to the Geoscience domain used by APE. In this domain, GIS tools do calculations. These calculations can be described by a GATA expression. When a tool is applied, all the operations in its GATA representation will be applied. Thus, a workflow represents a sequence of GATA expressions. As the tools of a workflow are applied in sequence, the expressions representing them are also applied in sequence. A workflow's GATA representation is the GATA expressions of its tools merged into a bigger expression. To be able to represent this we need to find all the possible ways to break a large expression into different smaller expressions. This has to be done in such a way that the original expression remains the same.

First, only one tool can be applied at the same time. Each time step or module state m_i can only contain one tool. A tool applies a complex GATA expression. Such expression consists of one or more GATA functions that are applied in order. These functions need to be represented in some form in a propositional formula.

I chose to represent the GATA expression by abstracting the tree representation. We view the GATA expression as a set of its functions. This allows the use of the tool states already in APE and thus reusing an already proven structure. This set then represents all the functions that are applied at the same time as the tool. This approach reduces the amount of possible workflows but does allow tools with an incorrect function order to be used. Therefore, it introduces the need for a post-process verification step.

With this chosen abstraction, I can now determine the possible ways to separate the GATA functions of the workflow specification into different tool states that would still result in a valid workflow. Between these states, the order of the GATA functions should represent the expression of the workflow specification. This will be demonstrated using the easy example from earlier. The example is repeated in Expression 8 using the parse tree of Figure 8.

The tree is parsed and built from top to bottom, but data flows from bottom to top. This means before *reify* can be applied, *pi* needs to be applied. However, a tool could exist that applies both *reify* and *pi*. This tool can only be applied if the *interpol* has been applied. The leaves, like *data* in this example, represent readily available input. We can ignore these data leaves when talking about the meaning of a graph as the data leaves are irrelevant for the function order. These constrains results in the function separation in states that can be found in Equations 9. In this representation, each state s_i represent a tool being applied. Tools are applied in order so s_0 is applied before s_1 and so on. Each row is a possible way to separate functions in tool states such that a workflow with such representation of GATA functions is considered correct.

$$reify\ (pi\ (interpol\ (input))) \tag{8}$$

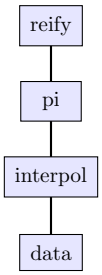

Figure 8: The parse tree representation of Expression 8, the simple example

$$\begin{aligned}
&s0 = \{interpol, pi, reify\} \\
&s0 = \{interpol, pi\} && s1 = \{reify\} \\
&s0 = \{interpol\} && s1 = \{pi, reify\} \\
&s0 = \{interpol\} && s1 = \{pi\} && s2 = \{reify\}
\end{aligned} \tag{9}$$

For this example, I introduce some mock-up tools to be part of the annotation set $A_{example}$. These tools are used to give a concise example of the constraints. The real examples are too verbose to easily show the principle. However, they are discussed later. The tools used are shown at 10. The annotation set example can also be found below.

$$\begin{aligned}
\text{reifyTool} &= reify(data) \\
\text{combinedTool} &= pi(interpol(data)) \\
\text{piTool} &= pi(data) \\
\text{interpolTool} &= interpol(data) \\
\text{unusedTool} &= revert(data)
\end{aligned} \tag{10}$$

$$\begin{aligned}
A_{example} = \{&(\text{reifyTool}, && (\{reify\}, && \{\})), \\
&(\text{combinedTool}, && (\{pi, interpol\}, && \{(pi, interpol)\})), \\
&(\text{piTool}, && (\{pi\}, && \{\})), \\
&(\text{interpolTool}, && (\{interpol\}, && \{\})), \\
&(\text{unusedTool}, && (\{revert\}, && \{\}))\}
\end{aligned} \tag{11}$$

Using this set of tools we can verify the possible workflows. There are only 2 possible workflows assuming type constraints of the tools themselves are valid. These can be found at Equation 12. Notice how both use a different set of tools. When we replace the tools by the sets of functions the tools represents, the workflows look like one of the state combinations found at 9.

$$
\begin{array}{lll}
s0 = \text{combinedTool} & s1 = \text{reifyTool} & \\
s0 = \text{interpolTool} & s1 = \text{piTool} & s2 = \text{reifyTool}
\end{array}
\tag{12}
$$

4.2 Constraints

APE takes its user input in the form of constraints. There are natural language templates that a user can fill in that represent SLTL formulas. These are translated into propositional formulas for each workflow length. When I want to add constraints to APE, I can define them in SLTL. Then translate these SLTL formulas into propositional formulas and let APE translate them into the format for the SAT solver.

Thus, I have a GATA expression in the workflow specification and tools annotated with a GATA expression as in the domain knowledge. I use these to generate a set of SLTL expressions that generates workflows as closely related to the described problem. These SLTL expressions have to ensure that the workflow can recreate the original expression. To accomplish this for our example, one of the state expressions from Formula 9 should hold. The order of functions of the original graph should be maintained. For our example input 3, this means one of the state equations (9) should hold.

The first step is to add constraints that relate tools to functions and functions to tools. This ensures neither can be present without the other. First I introduce the constraint that relates tools to functions. When a tool is present in a state, all the functions in its GATA expression must be present. The set of formulas that represent this is named *TA* (Tool Annotations). This set TA is expressed by Equation 13

$$
TA = \{\mathbf{F}(< t > true \rightarrow \bigwedge_{n \in V}^{V} < n > true) \mid (t, (V, E)) \in A\} \tag{13}
$$

$$
\begin{aligned}
TA_{example} = \{&\mathbf{F}(< \text{reifyTool} > true \ \rightarrow < reify > true),\\
&\mathbf{F}(< \text{combinedTool} > true \rightarrow < pi > true \wedge < interpol > true),\\
&\mathbf{F}(< \text{piTool} > true \rightarrow < pi > true),\\
&\mathbf{F}(< \text{interpolTool} > true \rightarrow < interpol > true),\\
&\mathbf{F}(< \text{unusedTool} > true \rightarrow < revert > true) \qquad \}
\end{aligned}
$$

The second constraint ensures functions are related to tools. When a function is present in a state, one of the tools that represent is must be present. This is represented by the following constraint set TU (Tool Usage) seen in Equation 14. It uses helper function TF that returns a set of tool names that have a specific function in their annotation which can be found in Equation 15

$$
TU = \{\mathbf{F}(< f > true \rightarrow \bigvee_{x \in TF(f)}^{TF(f)} < x > true) \mid f \in F\} \tag{14}
$$

$$
TF(f) = \{t \mid f \in V, (t, (V, E)) \in A\} \tag{15}
$$

$$
\begin{aligned}
TU_{example} = \{&\mathbf{F}(< reify > true \rightarrow < \text{reifyTool} > true),\\
&\mathbf{F}(< pi > true \rightarrow < \text{combinedTool} > true \vee < \text{piTool} > true),\\
&\mathbf{F}(< interpol > true \rightarrow < \text{interpolTool} > true),\\
&\mathbf{F}(< revert > true \rightarrow < \text{unusedTool} > true) \qquad \}
\end{aligned}
$$

These constraints create the environment to reason over the functions. A function relates to a one of the tools and a tool relates to a set of functions. This enforces tools to be used when functions are required to be present.

4.2.1 Input Constraints

As function and tool usage is enforced, the next constraints focus on the GATA expression of the workflow specification. The third constraint is function usage (FU). All tools present in the input should appear in the workflow. This is represented by the set FU as can be seen in Equation 16

$$FU = \{\mathbf{F}(< f > true) \mid f \in V_i \ \} \tag{16}$$

$$\begin{aligned} FU_{example} = \{&\mathbf{F}(< refiy > true), \\ &\mathbf{F}(< pi > true), \\ &\mathbf{F}(< interpol > true) \qquad\qquad \} \end{aligned}$$

The fourth constraint is regarding function prevention. The meaning of a workflow would change if a function is used that is not present in the original input. To realize this, I prevent functions not present in the input to appear in the workflows. This constraint is shown by Equation 17

$$FP = \{\mathbf{G}(\neg < f > true) \mid f \notin V_i,\ f \in F\} \tag{17}$$

$$FP_{example} = \{\mathbf{G}(\neg\ < revert > true)\}$$

The next set of constraints focus on the structure of the GATA expression. The fifth constraint ensures the order of the graph. This concerns the edges of the graph. As shown in Figure 6, *pi* needs to be calculated before or calculated with the same tool as *reify*. This holds for the whole graph so there is a relation for each edge. This leads to the order constraints (ORD) described by Equation 18.

$$ORD = \{\mathbf{F}((< e_2 > true \wedge < e_1 > true) \vee (< e_2 > true\ \wedge \mathbf{X}(< e_1 > true)) \mid (e_1, e_2) \in E_i\} \tag{18}$$

$$\begin{aligned} ORD_{example} = \{&\mathbf{F}((< interpol > true \wedge < pi > true) \vee (< interpol > true \wedge \mathbf{X}(< pi > true)), \\ &\mathbf{F}((< pi > true \wedge < reify > true) \vee (< pi > true\ \wedge \mathbf{X}(< reify > true)) \qquad \} \end{aligned}$$

We can limit the amount of results by enforcing the root of the input as the last function. Any workflow where the root function is not the last function in the workflow would not satisfy the input. The sixth constraint last function (LF) is shown by Equation 19.

$$LF = \{\mathbf{F}(< r > true) \wedge \mathbf{G}(\neg\ < r > true\ \vee \neg(\mathbf{X}\ \mathbf{X}(true))) \mid r \neq e_2,\ r \in V_i,\ (e_1, e_2) \in E_i\} \tag{19}$$

$$LF_{example} = \{\mathbf{F}(< reify > true) \wedge \mathbf{G}(\neg\ < reify > true\ \vee \neg(\mathbf{X}\ \mathbf{X}(true)))\}$$

The next constraints concerns the number of times a function can appear in a workflow. The seventh constraint enforces a function is used in exactly as many different tools as it appears in the input. This constraint is based on domain knowledge of Geoscience and on the tool examples. A tool uses a specific function only at most once. Therefore, when a function appears twice, tools that represent it should appear twice. for each function, the constraint depends on how often it appears in the graph. The constraint function count (FC) is shown by Equation 20. The helper function *generateFC* recursively generates the SLTL constraint for any length.

$$FC = \{generateFC(countFn(f_1), f_1) \mid f_1 \in V_i\} \tag{20}$$

$$\begin{aligned} countFn(f_1) =& count(\{f_2 \mid f_2 = f_1, f_2 \in V_i\}) \\ generateFC(0, f) =& \ true \\ generateFC(i, f) =& \ \mathbf{F}(< f > true \wedge \mathbf{X} \text{ generateFC}(i-1, f)) \end{aligned}$$

However, a drawback of this constraint is that it ignores some solutions. If a workflow specification needs a function to appear three times and two of those are in the same function, it would be excluded by this constraint. If a function is required three times it has to appear in the annotation of 3 different tool usages instances. Even with this drawback, I can still use this constraint. In the given tool set, there are no tools that have a function appear more than once. But if such a tool appears in the future this constraint needs to be adapted.

The final and eighth constraint also regards the amount of times a function can appear. This constraint limits the amount of time a function can appear. A function should not appear more times than it appears in the graph. This leads to the function limiting constraint (FL) that is shown by Equation 21. It reuses the *generateFC* function. It prevents any combination of functions longer the amount of times it appears in the input.

$$FL = \{\mathbf{G}(\neg generateFC(countFn(f_1) + 1, f_1)) \mid f_1 \in V_i\} \tag{21}$$

This constraint also has a drawback. If a workflow specification has a function that appears three times, one of those times can be a tool where the function is represented more then once. This can result in a solution that has three different instances of tools that represent that function but one in one of those tools the function appears more than once. Thus, allowing a solution with four appearances of a function.

These eighth constraints are generated from the GATA expression of the workflow specification and the additional tool annotations. The complete Gata Constraint Set (GCS) would look like Equation 22.

$$GCS = \{TA, TU, FU, FP, ORD, LF, FC, FL\} \tag{22}$$

The main goal of these constraints is to reduce the amount of possible solutions. It succeeds for our simple example $reify(pi(interpol(input)))$. However it does not yet capture functions that have more than one parameter.

4.2.2 Multiple Parameter Functions

The set of constraints from Equation 22 covers the GATA expressions that only have single parameter functions. However, functions are able to have more than a single parameter. Examples of this are the functions *lotopo* and *sigma*. This multiple parameter example can be seen in Equation 23 and Figure 9.

Before the function *lotopo* can be applied, both *pi* and *sigma* need to be applied. However, these functions are not dependent on each other. They are different parameters to the same instance of *lotopo* and thus different parts of the graph. Because of this, the order in which *pi* and *sigma* are applied does not matter. However, they have to be applied before or at the same time as *lotopo*. This behaviour changes the way the functions can be divided over states as can be seen in Equations 24.

$$lotopo(pi(interpol(data1)), sigma(data2, data3)) \tag{23}$$

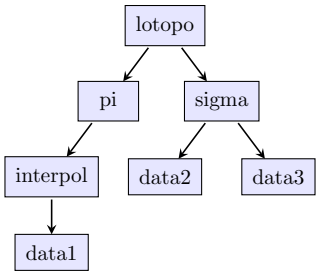

Figure 9: Graphical representation of the graph of Expression 23, the multiple parameter example

$$
\begin{array}{llll}
s0 = \{sigma, interpol, pi, lotopo\} & & & \\
s0 = \{interpol, pi\} & s1 = \{sigma, lotopo\} & & \\
s0 = \{sigma\} & s1 = \{interpol, pi, lotopo\} & & \\
s0 = \{sigma\} & s1 = \{interpol\} & s2 = \{pi, lotopo\} & \\
s0 = \{sigma\} & s1 = \{interpol, pi\} & s2 = \{lotopo\} & \\
s0 = \{interpol, pi\} & s1 = \{sigma\} & s2 = \{lotopo\} & \\
s0 = \{sigma\} & s1 = \{interpol\} & s2 = \{pi\} & s3 = \{lotopo\} \\
s0 = \{interpol\} & s1 = \{pi\} & s2 = \{sigma\} & s3 = \{lotopo\}
\end{array}
\tag{24}
$$

The original order constraint is not sufficient to capture this behaviour. For functions that have multiple parameters, there is no strict ordering. To loosen the strictness of this ordering, I revise the original constraint to look like equation ORD_{rev} which can be found in Equation 25.

$$
\begin{aligned}
ORD_{rev} = \{\{&\mathbf{F}((< e_2 > true \wedge < e_1 > true) \vee \\
&(< e_2 > true \ \wedge \mathbf{X}(< e_1 > true)) && \mid e_2 = e_3,\ (e_1, e_3) \in E_i,\ (e_1, e_2) \in E_i\}, \\
\{&\mathbf{F}(< e_2 > true \ \wedge \mathbf{F}(< e_1 > true)) && \mid e_2 \neq e_3,\ (e_1, e_3) \in E_i,\ (e_1, e_2) \in E_i\}\}
\end{aligned}
\tag{25}
$$

$$
\begin{aligned}
ORD_{example2} = \{&\mathbf{F}(< pi > true \ \wedge \mathbf{F} \ < lotopo > true), \\
&\mathbf{F}(< sigma > true \ \wedge \mathbf{F} \ < lotopo > true), \\
&\mathbf{F}((< interpol > true \wedge < pi > true) \vee (< interpol > true \wedge \mathbf{X}(< pi > true))\}
\end{aligned}
$$

The usage of **F** captures the both the order and looseness required by multiple parameter functions. Both *pi* and *sigma* can be used at the same time as *lotopo*. If *lotopo* is not used at the same time, it has to appear somewhere after. However with this solution, a workflow would still be valid when *lotopo* is applied before either *pi* or *sigma* and it is applied again afterwards. This is not a problem. It is solved by either the maximum usage of the function constraint or the filtering of the solutions afterwards. The constraint does lower the amount solutions while preserving the new situations. This leads to changing the constraint set to Equation 26.

$$
GCS_{rev} = \{TA, TU, FU, FP, ORD_{rev}, LF, FC, FL\} \tag{26}
$$

4.3 Conversion Tools

So far, I have talked about how to handle tools that do calculations. These tools have a representation in GATA. Using formulas to link the GATA representation to tools allowed to reason over the gata functions instead. This section discusses how to handle tools without GATA annotation. These tools do not do Geoscience relevant calculations. They handle data transformation instead. An example would transforming in a .csv file into a geographical data type or transforming one data type to another without changing the actual data. These tools do not directly impact the meaning of the workflow and the GATA structure, but including these tools will help a user find a workflow for their current situation. A user might not have the data in a format required for a strict workflow. In this case, a user will be able to pick an extended workflow that could handle the necessary transformation. I discuss the implementation using the GATA constraint set from Equation 26.

Now that I have discussed the relevance of conversion tools, I can discuss how to include them in the environment. A conversion tool needs to have a normal APE tool annotation. This way, APE includes it in the normal tool constraints. It will make sure its usage will be type correct, just like it would with any other tool. For simple inclusion, it is required that it does not have a GATA annotation. Without an annotation, the tool is not prevented from usage by the FP constraint and usage is not forced by the FU constraint. Thus, without GATA annotation, a tool is not forced in a workflow by the GATA constraints, but it can always be added. This freedom allows APE to add conversion tools if the types require it.

This is enough for the first use case of a tool, where a tool read a file and load a GIS data structure. In this case, a tool needs to be put in front of a branch of the GATA structure. If we give such a tool the special function *conversion*, we can see how it would fit in the tree. Figures 10 and 11 show examples where data needs conversion before a function.

In these figures, you can see the original graph is still the same. There is an empty node added but the original edges are untouched. The ORD_{rev} constraint does not need to change to allow this behaviour as the edges are unchanged. Thus a tool that needs to be applied before the first tool of a branch of the input does not cause conflicts with the current constraints.

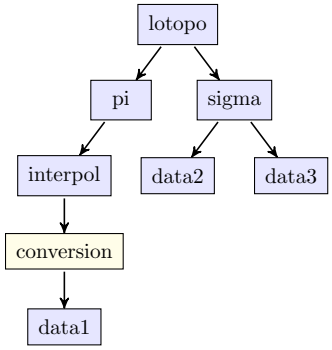

Figure 10: Graphical representation of the graph of the multiple parameter example. A conversion tool is used to convert *data1* to input of *interpol*

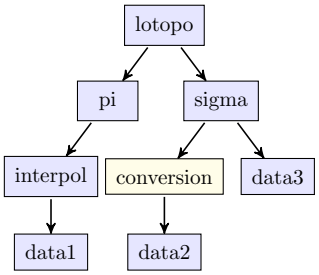

Figure 11: Graphical representation of the graph of the multiple parameter example. A conversion tool is used to convert *data2* to input of *sigma*

The last use case is when a tool converts data types in between other functions. Figures 12 and13 are examples of this case. Here, the edges of the original graph are changed. In Figure 12 the edge $(pi, interpol)$ is broken into 2 different edges with *conversion* in between. This also occurs in Figure 13 where the edge *(lotopo,sigma)* is broken in 2 edges. These examples represent edges broken up from both single and multi parameter functions. Both cases are represented differently in ORD_{rev} and should be taken into account

differently.

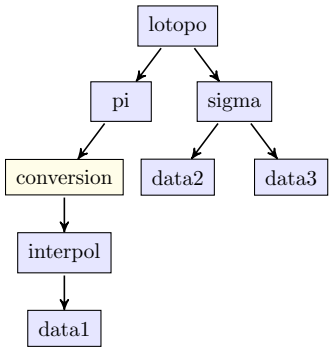

Figure 12: Graphical representation of the graph of the multiple parameter example. A conversion tool is used to convert *data1* to input of *interpol*

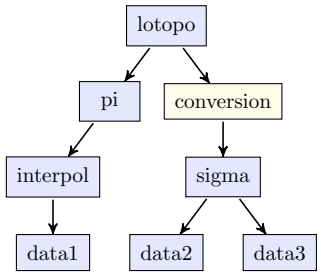

Figure 13: Graphical representation of the graph of the multiple parameter example. A conversion tool is used to convert *data2* to input of *sigma*

The single parameter function is represented by Figure 12. To allow this behaviour, we need to look at the constraint of the constraint of the edge. For the specific edge of $(pi, interpol)$ the constraint looks like $F((< interpol > true \wedge < pi > true) \vee (< interpol > true \ \wedge X(< pi > true))$. This constraint enforces *pi* and *interpol* in the same state or *interpol* in one state and *pi* in state directly after it. However, if we want to allow a conversion tool in between these tools we also need to allow the specific case that $s0$ constains *interpol*, that $s1$ contains a conversion tool and that $s2$ containts *pi*. If we take this into account the constraint for this specific edge looks like $\mathbf{F}((< interpol > true \wedge < pi > true) \vee$
$(< interpol > true \wedge \mathbf{X}(< pi > true)) \vee (< interpol > true \wedge \mathbf{X}(< conversion > true \wedge \mathbf{X}\ \mathbf{X}(< pi > true))))$.
This can be generalised and be used to revise the ORD_{rev} constraint. The new constraint ORD_{conv} can be found in Equation 27.

$$
\begin{aligned}
ORD_{conv} = \{\{&\mathbf{F}((< e_2 > true \wedge < e_1 > true) \vee \\
&(< e_2 > true \ \wedge \mathbf{X}(< e_1 > true)) \vee \\
&(< e_2 > true \ \wedge \mathbf{X}(< conversion > true \wedge \\
&\qquad \mathbf{X}(< e_1 > true)))) && \mid e_2 = e_3,\ (e_1, e_3) \in E_i,\ (e_1, e_2) \in E_i\ \}, \\
&\{\mathbf{F}(< e_2 > true \wedge \mathbf{F} < e_1 > true) && \mid e_2 \neq e_3,\ (e_1, e_3) \in E_i,\ (e_1, e_2) \in E_i\}\}
\end{aligned}
\tag{27}
$$

However, for this constraint to be able to work, conversion tools need to be annotated with a special tag. Otherwise I can't enforce that only conversion tools can be allowed using the ORD_{conv} constraint. This *conversion* tag needs to be included with the TA and TU constraints. Otherwise *conversion* can be true without a conversion tool being used. Furthermore, the *conversion* tag can't be included in the FP constraint. As *conversion* is never in the input, it would always be excluded. Thus, FP needs to be adapted to FP_{conv} which can be found in Equation 28.

$$
FP_{conv} = \{\mathbf{G}(\neg(< f > true)) \mid f \neq conversion, f \notin V_i,\ f \in F\} \tag{28}
$$

The multiple parameter function is represented by Figure 13. Like with the single parameter function, I need to look at the edge and how it is represented in the constraint. For the edge of $(lotopo, sigma)$, the constraint looks like $\mathbf{F}(< sigma > true \ \wedge \mathbf{F}(< lotopo > true))$. This constraint already includes the additional case of allowing the conversion tool. Thus, for the multiple parameter functions we do not need to revise constraints.

As I have shown, with minimal changes to the constraints I can allow conversion tools. Workflows will include these tools if the types of the tools in the workflow allow it.

4.4 Graph Comparing and filtering

Using the GATA input to generate these constraints reduces the amount of possible workflows APE is able to generate. However the constraints are not exactly compliant with the workflow specification. There might be some workflows in the resulting solutions that do not represent the input correctly. There is also the issue that in the tool annotation, the ordering of the functions is discarded. This might lead to incorrect tools appearing in the resulting workflow. An of such tools example would be tools where the correct functions are present but are in the wrong order. Thus those tools represent a different GATA expression. This is an inherent property of the chosen abstraction and it forces us to verify the solutions APE returns.

To filter solutions we need to be able to recreate a complex GATA expression from a workflow and to be able to compare the tree structure of two expressions. Comparing two trees is a relatively trivial task. You take the root of both trees and traverse to both of them in lockstep, then compare the nodes from both trees. When one pair does not match, the trees are not equivalent. When comparing nodes, only the nodes that hold function names should be considered. The names of the data leaves are not required to be the same. These hold just a name for the relevant data.

To recreate a complex expression we need to look at a workflow. The root of the expression is always applied by the last tool in the workflow. A function can only be applied when its parameters are applied. Thus, if we try to recreate the expression I can start from the root, the first function and the last tool.

I use Equation 29 from the simple example to demonstrate this principle. The GATA expression used as input is $reify(pi(interpol(input)))$ and the domain is the tool annotations from Equation 13. It shows a simple workflow of two tools, first it shows the use of combinedTool then use reifyTool.

$$s0 = \text{combinedTool} \qquad s1 = \text{reifyTool} \tag{29}$$

Figure 14: Visualisation of the example workflow from Equation 29

To recreate the original parse tree we need to start at the outer most function. In a workflow, this is always the last tool. In this case that is reifyTool. Then take the representation of the second to last tool (child tree) and combine this with the representation of the last tool (parent tree). To combine these trees, I take one leaf of the parent tree and replace this with the root of the child tree. The result of this operation can be seen in Figure 15.

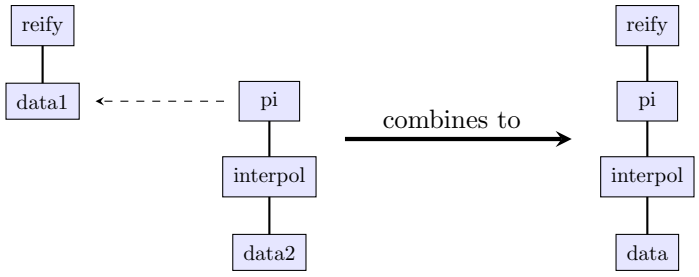

Figure 15: Application of one tool representation to another to create the more complex expression $reify(pi(interpol(data)))$

This is a simple example that only consists of two tools that have single parameter GATA representations. Usually, workflows consist of more than two tools. In this case, I repeat this process from the last tool to the

first tool in the workflows, adding each new GATA representation to the accumulated graph. This creates a combined expression that represents the workflow. When this expression is equivalent to the specification expression, the workflow is a valid workflow for that specification.

However, The process is more complex when GATA expressions consist of multi parameter expressions. This increases the amount of possible combinations. If there is a function with 2 parameters, all subsequent functions have 2 possible leaves to substitute. In a situation where in a workflow of 3 tools the last tool has 2 parameters, the amount of possible solutions for this example is 2^2. This is the amount of parameters to the power of the amount of subsequent tools left.

Each of these solutions needs to be verified until a matching solution has been found. If no matching solution can be found, the workflow is invalid and should not be considered a solution. Thus multi parameter functions exponentially increase the amount of solutions needed to be verified. However, this method guarantees only valid workflows are returned as solutions.

5 Comparison Between APE and GATA-adapted APE

This section compares using the constraints available in APE to the constraints generated by a GATA input. First the used examples are explained followed by a metric evaluation discussion of both approaches. Finally both approaches are compared conceptually.

5.1 Examples

For the evaluation, I use four different examples. These examples have been prepared by Simon Scheider. The examples are increasingly complex. I give an explanation, the GATA representation and the APE constraints of each example. For each example an example GATA expression and workflow have been prepared.

5.1.1 Example 1: Noise Portion

This is the first and most simple example. This example represents the question "Where in Amsterdam is noise more intense than 70 dB?". The target workflow consists of 5 tools and can be seen in Figure 16. The given GATA expression can be seen below. The tools relevant to this example can be seen in Table 3.

```
reify(
    pi1 (
        sigmale(
            bowtie(
                revert(contour_noise),
                deify(
                    merge(
                        pi2(
                            sigmae(objectregions_muni, object_utrect)
                        )
                    )
                )
            ),
            70
        )
    )
)
```

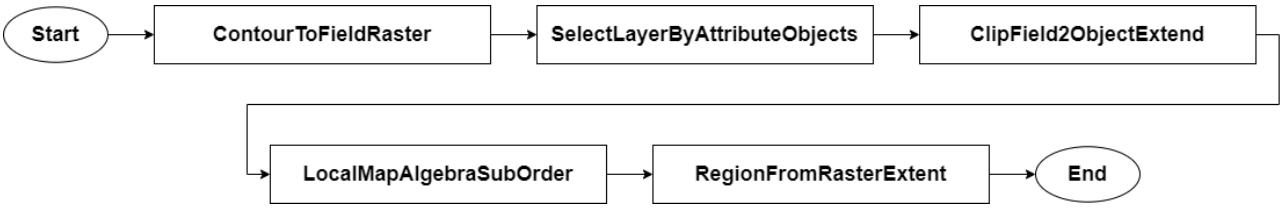

Figure 16: The given workflow for the noise portion example

Tool name	GATA expression
ContourToFieldRaster	$revert(contour\_x)$
SelectLayerByAttributeObjects	$sigmae(objectregions\_x, object\_y)$
ClipField2ObjectExtent	$bowtie(field\_x, deify(merge(pi2(objectregions\_x))))$
LocalMapAlgebraSubOrder	$sigmale(field\_x, ordinal\_y)$
RegionFromRasterExtent	$reify(pi1(field\_x))$

Table 3: Tools required for the noise portion example

5.1.2 Example 2: Noise Proportion

The second example has the longest resulting workflows. All the solutions have 7 tools in their workflow. Figure 17 is an example of this. The question this example tries to solve is "What is the proportion of the region where noise is greater than 70 dB in Amsterdam?". The GATA expression that calculates this can be seen below. The tools relevant can be found in Table 4. A possible workflow can be seen in Figure 17. The GATA expression branches high in the parse tree. This allows for a lot of variation when a part of the expression is represented in the workflow.

```
bowtie_ratio (
    groupby_size (
        sigmae (
            lotopo (
                pi1 (
                    sigmale(revert(contour_noise), 70)
                ),
                sigmae(objectregions_muni, object_Amsterdam)
            ),
            in
        )
    ),
    groupby_size (
        sigmae (
            lotopo (
                deify (
                    merge (
                        pi2 (objectregions_muni)
                    )
                ),
                sigmae(objectregions_muni, object_Amsterdam)
            ),
            in)))
```

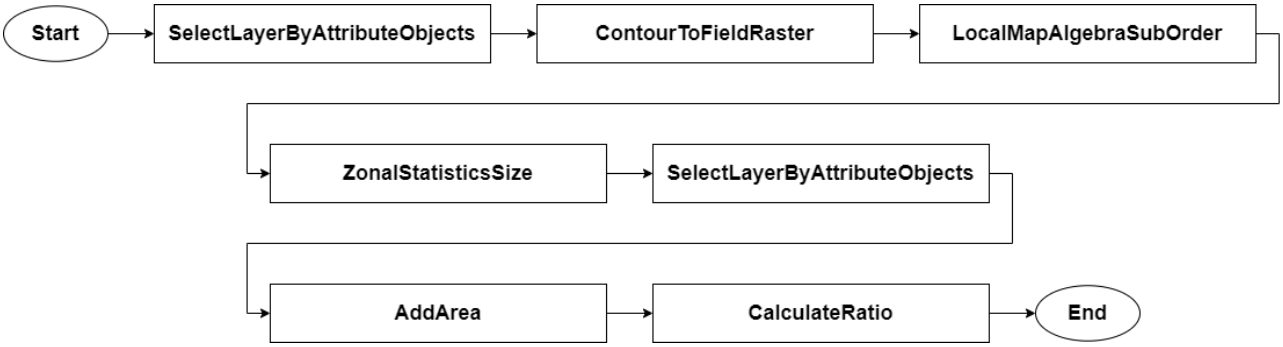

Figure 17: The given workflow for the noise proportion example

Tool name	GATA expression
ContourToFieldRaster	$revert(contour\_x)$
SelectLayerByAttributeObjects	$sigmae(objectregions\_x, object\_y)$
LocalMapAlgebraSubOrder	$sigmale(field\_x, ordinal\_y)$
ZonalStatisticsSize	$groupby\_size(sigmae(lotopo(pi1(field\_x), objectregions\_y), in))$
AddArea	$groupby\_size(sigmae(lotopo(deify(merge(pi2(objectregions\_x))), objectregions\_x), in))$
CalculateRatio	$bowtie\_ratio(objects\_x, objects\_y)$

Table 4: Tools required for the noise proportion example

5.1.3 Example 3: Object Amount and Distribution

The third example has the shortest workflow. The GATA expression is relatively complex for solutions with only 3 tools as can be seen in Figure 18. The question this example tries to solve is "What is the number of people for each neighborhood in Utrecht?". The GATA expression that calculates this can be seen below. The tools required can be found in Table 5 .

```
groupby_sum (
    bowtie* (
        sigmae (
            otopo (
                objectregions_households,
                bowtie (
                    objectregions_neighbourhoods,
                    pi1 (
                        sigmae (
                            otopo (
                                objectregions_neighbourhoods,
                                sigmae (objectregions_muni, object_utrecht)
                            ),
                            in
```

```
                    )
                )
            )
        ),
        in
    ),
    objectcount_households
  )
)
```

Figure 18: The given workflow for the object amount and distribution example

Tool name	GATA expression
SelectLayerByAttributeObjects	$sigmae(objectregions\_x, object\_y)$
SelectLayerByLocationObjects	$bowtie(objectregions\_x, pi1(sigmae(otopo(objectregions\_x, objectregions\_y), in)))$
SpatialJoinSumTessCount	$groupby\_sum(bowtie * (sigmae(otopo(objectregions\_x, objectregions\_y), in), objectcounts\_z))$

Table 5: Tools required for the object amount and distribution example

5.1.4 Example 4: Field Amount and Distribution

The final example has the most interesting dynamics. The question this examples tries to solve is "What is the average temperature for each neighborhood in Utrecht" The GATA expression that calculates this can be seen below. The tools required can be found in Table 6. This example includes a conversion tool. The conversion tool is a necessity in concrete workflows. However, it bloats the amount of possible workflows by repeated appeared as it is not bound by constraints. But including it is a necessity as it is required in concrete workflows. The example also shows that tools with the same GATA representation can be used interchangeably. Figures (19,20) show both example workflows.

```
groupby_avg (
    bowtie* (
        sigmae (
            lotopo (
                pi1 (field_x),
                bowtie (
                    objectregions_x,
                    pi1 (
                        sigmae (
                                otopo (
                                    objectregions_x,
                                    sigmae (objectregions_x, object_y)
                                ),
```

```
                    in
                )
            )
        )
    ),
    in
),
interpol (
    pointmeasures_x,
    deify (merge (pi2 (objectregions_y)))
)
)
)
```

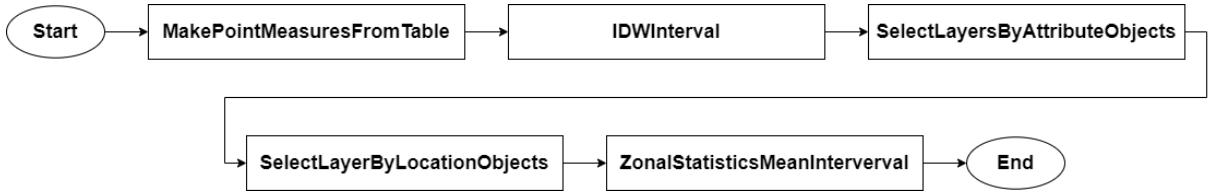

Figure 19: One of the given workflow for the Object field and distribution example

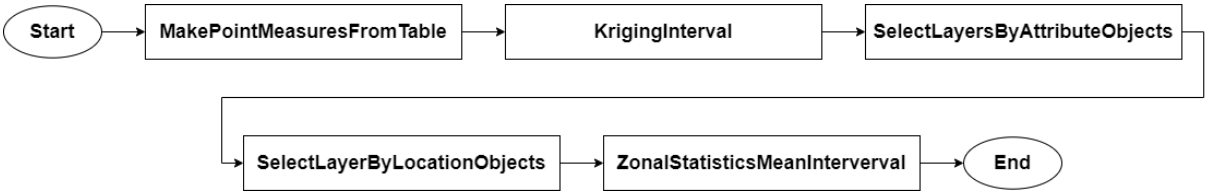

Figure 20: One of the given workflow for the Object field and distribution example

Tool name	GATA expression
MakePointMeasuresFromTable	$conversion$
SelectLayerByAttributeObjects	$sigmae(objectregions\_x, object\_y)$
SelectLayerByLocationObjects	$bowtie(objectregions\_x, pi1(sigmae(otopo(objectregions\_x, objectregions\_y), in)))$
IDWInterval	$interpol(pointmeasures\_x, deify(merge(pi2(objectregions\_y))))$
KrigingInterval	$interpol(pointmeasures\_x, deify(merge(pi2(objectregions\_y))))$
ZonalStatisticsMeanInterval	$groupby\_avg(bowtie * (sigmae(lotopo(pi1(field\_x), objectregions\_y), in), field\_x))$

Table 6: Tools required for the Field amount and distribution example

5.2 Metric evaluation

This section evaluates the results of running APE with the GATA constraints. First, I discuss the GATA-adapted APE results and later compare these to the runs with APE. These tests are done on a Lenovo

Y50 laptop with an Intel i7-4710HQ 8 core processor that runs at 2.5GHz. It has 8 GB of RAM and runs Windows 10 64bits. The data in the run time graphs is average of 5 runs.

The evaluation runs use APE configuration files with same the relevant parameters. The maximum amount of solutions that we allow is set to 1000. It takes a lot of effort for a user to already compare 1000 solutions and pick the best one. Thus generating more than 1000 would not be relevant. Usually the best solutions are among the shortest solutions. From testing, Generating 1000 solutions is enough to find all solutions for the shortest two solution lengths for a specific use case. For the maximum solution length, I use a max length of 9. This length is longer than the given solutions. I compare the run time for length 9 to the run time of length 10 separately.

5.2.1 Amount of Solutions

In this section, I evaluate the amount of solutions each method generate per solution length. This tells us how easy it will be for the user to pick a solution to use. The amount of solutions generated using the GATA constraints are the amount after the graph filtering. For each example I have prepared graphs showing only the GATA solutions and graph showing GATA and APE in the same graph.

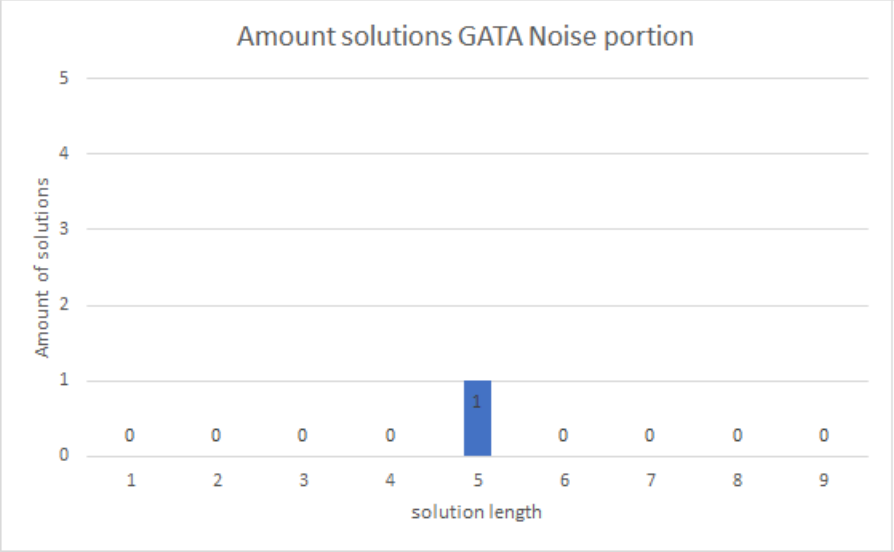

Figure 21: The amount of solutions for each workflow length for Example 1, the noise portion example

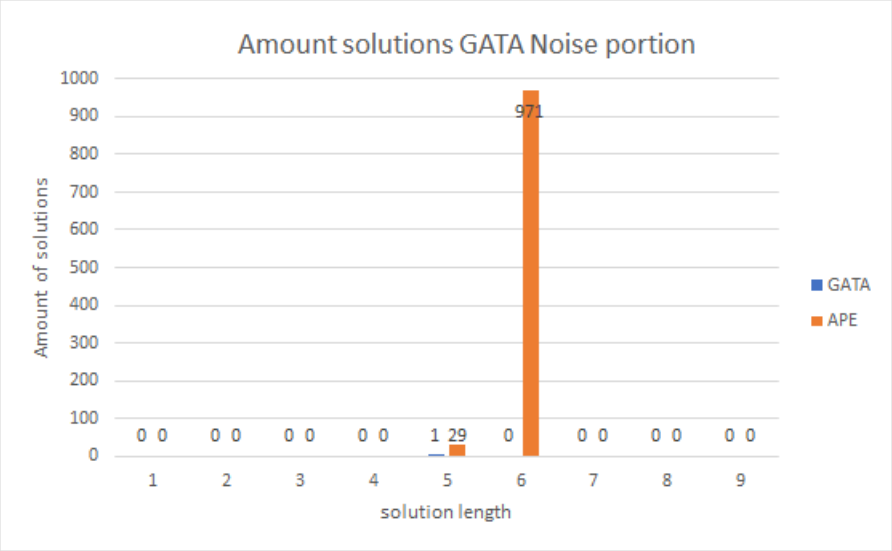

Figure 22: A comparison between the amount of solutions GATA and APE constraints generate for each workflow length for Example 1, the noise portion example

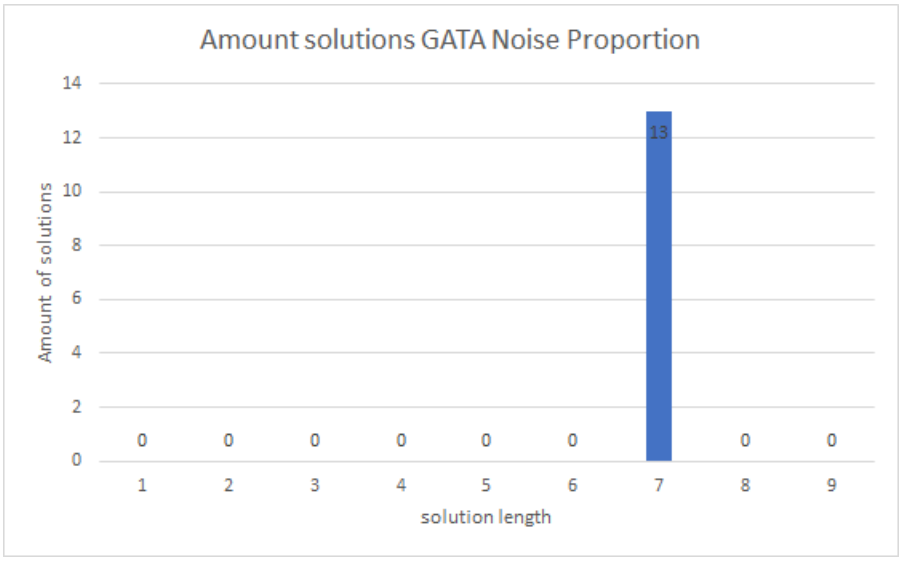

Figure 23: The amount of solutions for each workflow length for Example 2, the noise proportion example

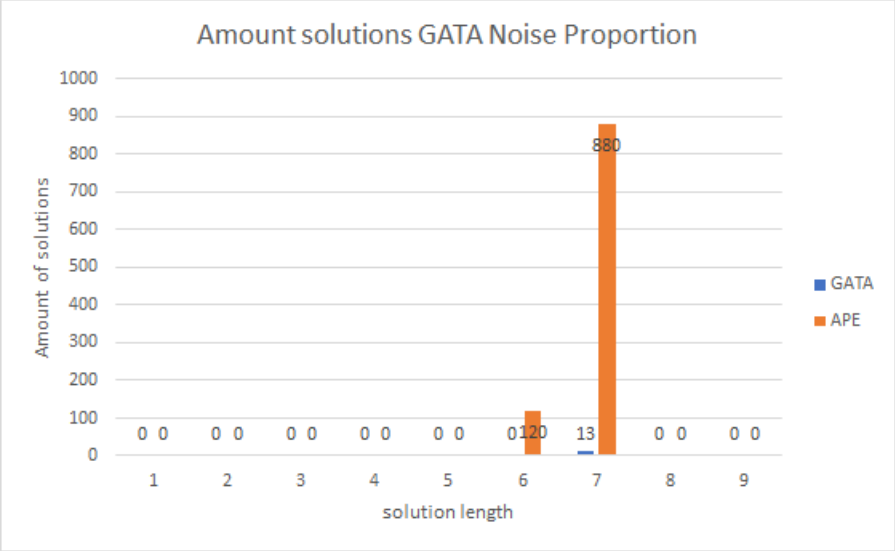

Figure 24: A comparison between the amount of solutions GATA and APE constraints generate for each workflow length for Example 2, the noise proportion example

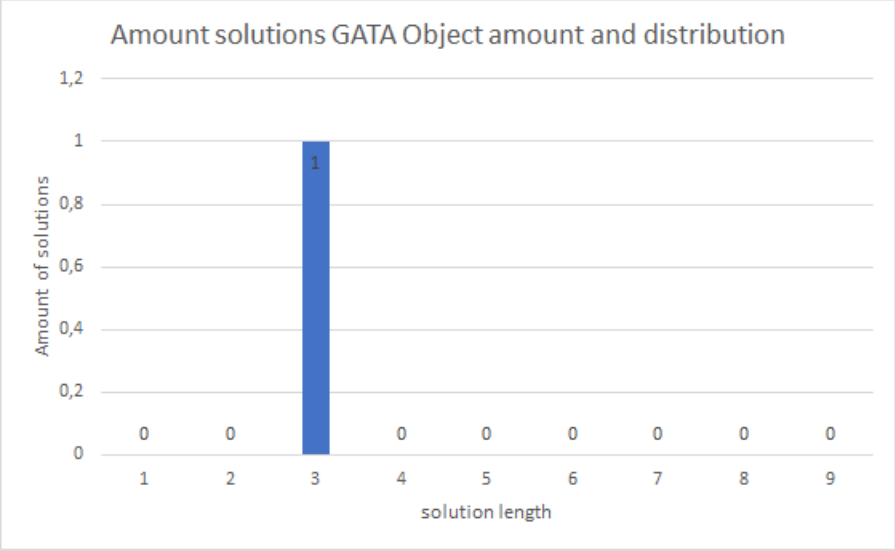

Figure 25: The amount of solutions for each workflow length for Example 3, the object amount example

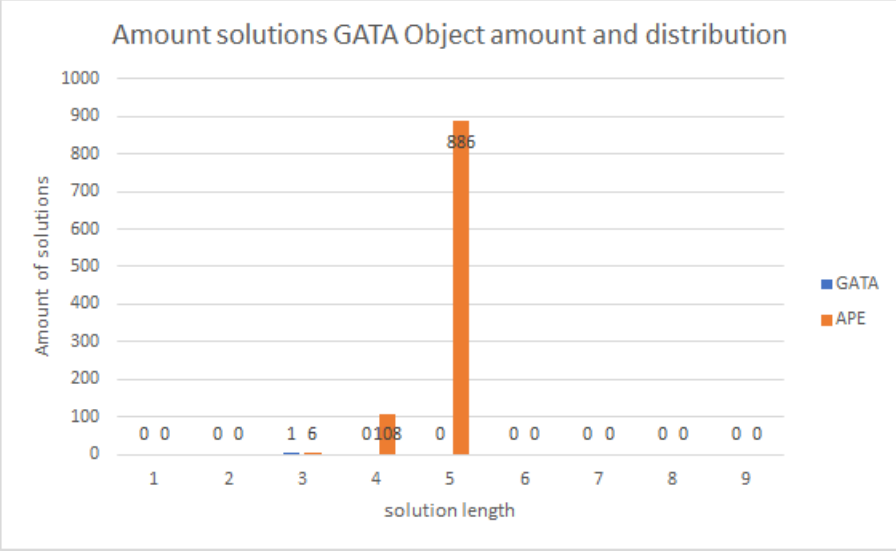

Figure 26: A comparison between the amount of solutions GATA and APE constraints generate for each workflow length for Example 3, the object amount example

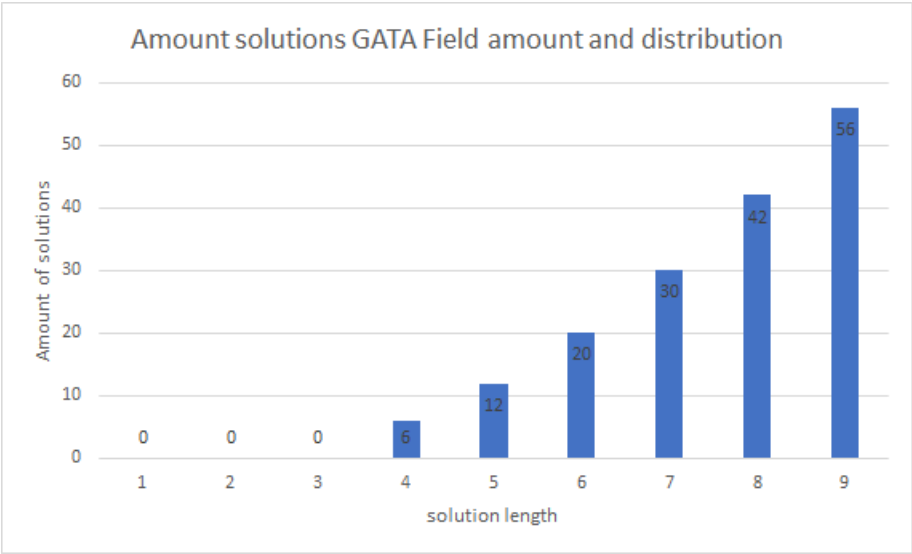

Figure 27: The amount of solutions for each workflow length for Example 4, the field amount example

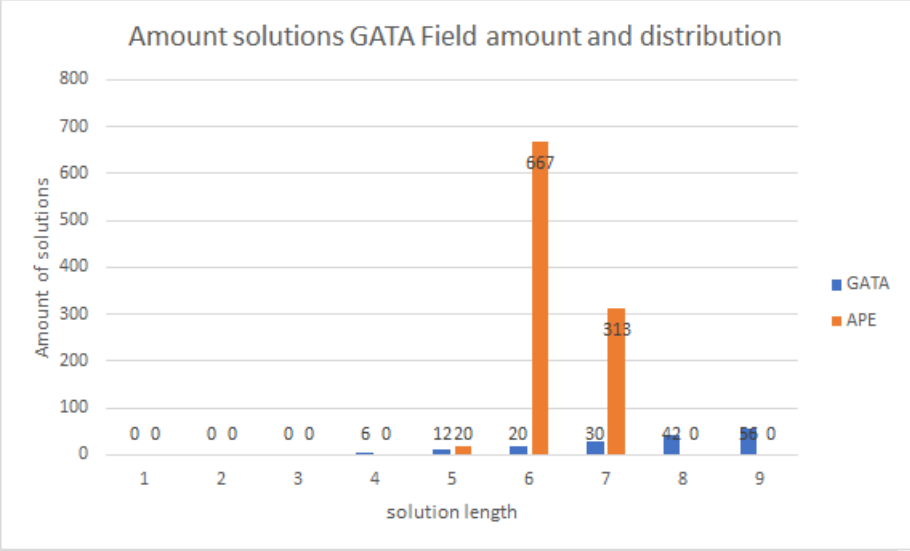

Figure 28: A comparison between the amount of solutions GATA and APE constraints generate for each workflow length for Example 4, the field amount example

The first thing that is remarkable, is that the amount of solutions for GATA is quite low. Example 1, the noise portion example, shows only one results. The noise portion example has a small GATA expression with specific tools in the domain. This shows, that with the GATA constraints, it finds the specific workflow that the user would be looking for.

Example 2, the noise proportion example, has 13 solutions. This example has more solutions as the GATA branches early in the parse tree. This means two entirely separate calculations need to be done before the final tool. The order only needs to be taking into account for each branch of the GATA expression. This means interleaving of the tools of these branches can happen and result in multiple correct workflows. This example shows this by having 13 solutions.

Example 3, the object amount example, also only has one solution. This example has a complex GATA but yields a solution that contain three tools. This shows the GATA constraints also can yield simple workflows when the domain allows it. But as the workflow contains a small amount of tools, there is no variation allowed in the solution.

Example 4, the field amount example, has workflows of all lengths of four and longer. This happens as the types of the tools of the required workflow allow for a conversion tool to be included. This explains the variation in solution length. At length four, the tool is not included. At every subsequent length, the tool is included one or more times. Another thing to note is that for the length of 4 is there are 6 solutions. But for either example workflow of Figure 19 or Figure 20 there are only 3 ways to keep the tool order as the GATA expression. This shows that both examples are found in the results.

When comparing the Solutions from APE and GATA, we make a two of observations. The first observation is that the APE runs always reach the maximum of 1000 results before reaching the maximum solution length. The second is that the APE runs always have solutions in multiple solution lengths. This shows that GATA constraints are more domain-specific in comparison to APE.

5.2.2 Run Time Evaluation

Next I evaluate the run time. In this section, I compare the APE synthesis run time of both approaches. The graph filtering will be evaluated next section.

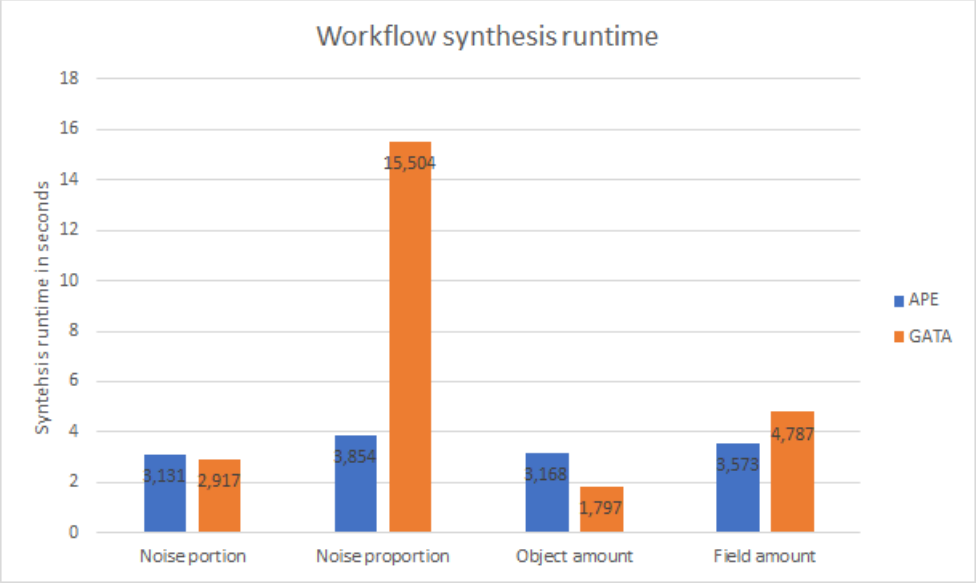

Figure 29: An overview of the run time for the synthesis portion of both approaches. Compared per example

An overview of the synthesis run time per example can be found in Figure 29. The first thing I notice is that the run time for Example 2 ,the noise proportion example, is a couple times longer longer than the other run times. The run time for the GATA examples is mostly determined by the time it takes to convert the GATA expressions into constraints specific for each solution length. This is the reason there is such a big of a difference between run times of examples.

The APE run times are all closer to each other. The limiting factor for APE is the amount of solutions the synthesis will generate. The APE constraints used are much less restrictive for these examples. Thus the synthesis algorithm generates the maximum amount of solutions.

The constraint conversion into CNF is the biggest bottleneck for the GATA approach. The longer the solutions are, the longer the translation time into CNF. This is especially evident when comparing the problem set up time for the solution length of 9 and 10. This can be seen in Figure 30.

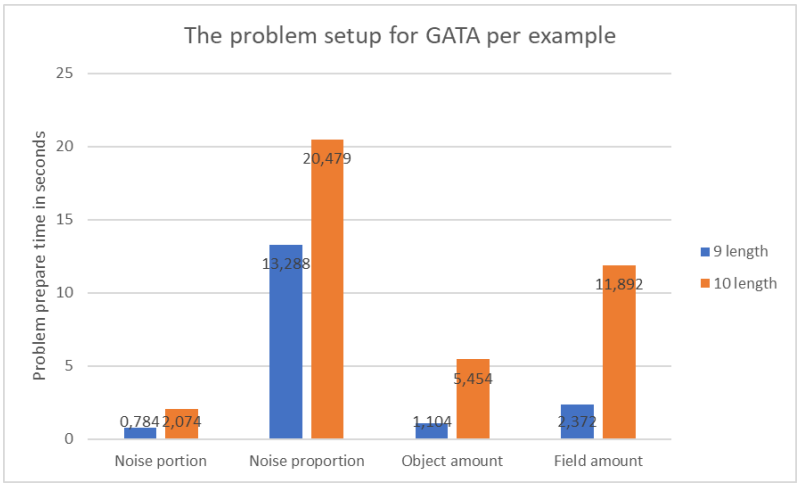

Figure 30: An overview of the problem set using the GATA constraints for solution length 9 and 10

5.2.3 Graph Filter Evaluation

In this section, I evaluate the graph filtering part of the GATA constraint part. This post-process step ensures that all solutions comply with the specification expression. As a trade-off, there is some time spent recreating and comparing the graphs, but the amount of solutions it reduces is quite significant. This can be seen in Figure 31.

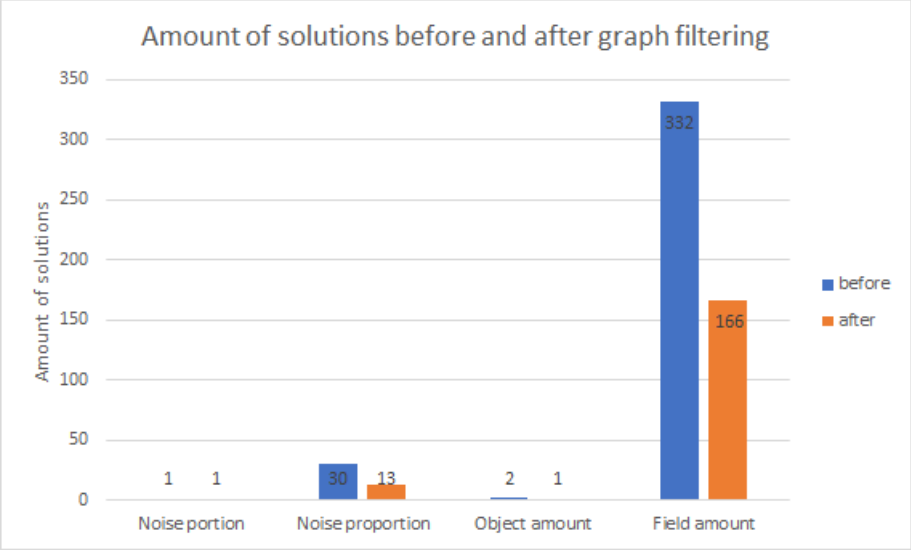

Figure 31: The amount of solutions before and after the graph filter portion of the GATA constraints.

The impact on Example 1 and Example 3 is minimal as the amount of solutions was already small. For the other examples, it reduces the amount of solutions by about half. Example 2, The noise proportion example, has some tool orders that do not comply as the solution length is quite long. Example 4 ,the field amount example, has the most solutions removed as it also has the most solutions generated. In these examples the filtering definitely has an impact. as it removes a lot of solutions that do not comply with the specification.

For the time efficiency, we need to look at the run time for the graph filtering. This can be found in Figure 32. For most examples the run time is below half a second. These are times that would not impact the general run time much. However, the run time for Example 2, the noise proportion example, is more than 5 seconds. This is quite long. It is not unacceptable for the guarantee that the resulting solutions comply with the specification. The naive approach to the verification of solutions shows in the run time of this example.

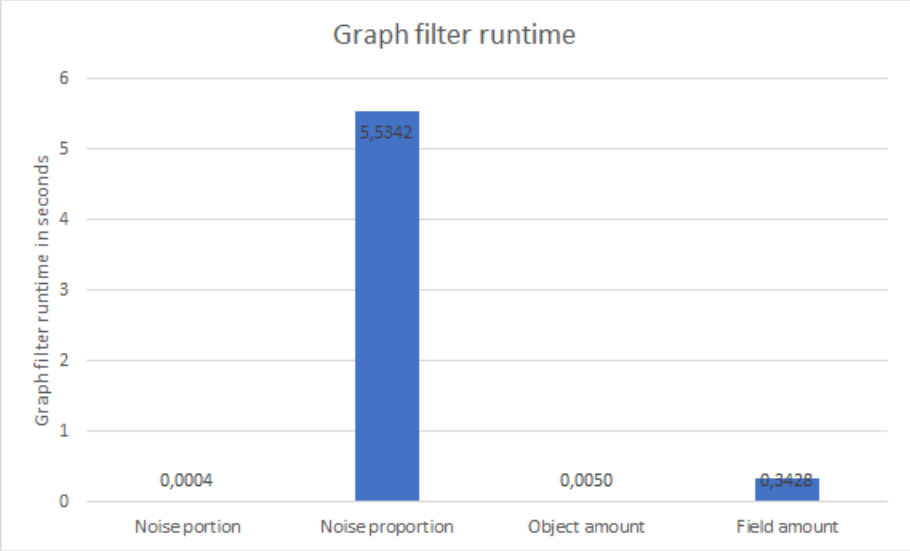

Figure 32: The run time for the graph filter portion of the GATA constraints displayed for each example.

5.3 Time Complexity And Correctness

In this section, I briefly touch on the time complexity and correctness of this approach. The correctness regards the constraints and the post-process step individually. The time complexity discussion touches on the transformation from SLTL to a propositional formula and the graph recreation algorithm .

The constraints are based on abstracted versions of GATA expressions. The GATA expressions that annotate the tools are abstracted. Thus the constraints cannot guarantee a perfect representation. The constraints try to approach the GATA expression of the workflow specification. This allows solutions that have a different GATA representation than the workflow specification to be in the solution set. The post-process step guarantees the solutions are correct and filters the incorrect solutions.

The post-process step recreates a GATA expression of a workflow and compares it to the GATA expression of the workflow specification. The recreation algorithm creates the expression representing the workflow by combining the expressions of tools. It starts by the last expression and adds the previous expression on one of the leaves of the parse tree of that expression. This results in multiple bigger GATA expressions that a workflow can represent. Then each of these expression is compared against the specification expression. The parse trees of a candidate expression and the specification expression are walked together in lock step. If a match cannot be found, the candidate expression is discarded and if there is a match, the workflow matches the specification. By verifying all expressions of a workflow, a match with the original expression will be found if it exists and thus it suggests correctness.

The time complexity of generating the constraints depends on how the SLTL formula's are translated to propositional formulas. It depends on how many **G** and **F** operators are in a formula and on the solution length. Each of these operators increase the amount of propositions in the resulting formula exponential. The amount of these operators in a formula is called p and the solution length n. Then the generation of the propositional formulas has a time complexity of $\mathcal{O}(n^p)$. This complexity stems forth from the conversion method from SLTL to propositional formula. The **F** and **G** operators list all possible situations from state 0 to state n. This shows that SLTL formulas with long **F** and **G** chains are inefficient. However there are two constraints that do use these operators chained together, but these limit the possible workflows so much that it is reasonable to use these constraints.

The graph filter algorithm of the post-process step uses the most naive process to recreate the graph. It creates all possible combinations. In the easiest case all tools have no function in their representation that have multiple parameters. In this case there is only one solution. However, most tools have one or more functions in their representation with two parameters. Each such function adds a possible way to combine two tools. Thus, if a representation has k functions with two parameters, it adds k possible ways to combine two tools. If two tools have k functions, it adds k^2 possible solutions. Then if a solution has n tools with k functions for each GATA representation, there are $(n-1)^k$ solutions. This leads to a worst possible time complexity of $\mathcal{O}(n^k)$. This can be seen in the graph filter run time in Figure 32. Example 2, the noise proportion example, has many more functions with two parameters and that drastically increases the time to post-process this example.

However, for the other use cases, the naive approach was not a problem as either their solutions where short or their GATA expressions contained few repeating function and few multiple parameter functions. Repeating functions create long sequences of **F** and **G** operators and Short solutions and few parameter functions reduce the amount of possible graph combinations.

5.4 Approach Comparison

This section compares both approaches semantically. The approaches try to accomplish the same goal but in different ways. The GATA constraints build upon APE, the way the domain knowledge is handled is the same. This GATA-adapted APE relies on the property of APE to combine tools in a correct way. Only the way the user constraints encoded is different.

A user of APE specifies the input and output of their workflows. In addition to this, the user specifies some constraints. These either reason over a specific tool or an abstract tool. By using abstract tools the user can loosely specify what kind of behaviour they seek. However, these constraints are domain dependent. A user can only reuse their constraints and taxonomies if these are designed in a general abstract way. If these are not designed as such, a user might have to respecify their constraints.

This is in contrast to using GATA as constraints. This generality is inherent to the use of GATA as constraints. A GATA expression precisely specifies the required operations. This expression is used to find workflows that explicitly match that GATA expression of the workflow specification. The user expresses their intent using an independent language instead of the domain directly.

However, an important difference with these approaches, is that all the solutions APE returns adhere exactly to the constraints of the workflow specification. The GATA-based approach translates the GATA expression of the workflow specification to a set of constraints that approximate the required solutions. Without the post-process step that compares solutions to the specification, there would be incorrect solutions in the final list.

In addition to these general differences there are also some differences in the solutions each approach generates. As seen in the previous section, the GATA-based approach generates less solutions in general. It is more restricted in the solutions it can generate. Especially for solutions longer than the minimum length that generates solutions. APE has a lot of possibilities for repeating tools.

The GATA approach limits this by having a maximum amount of times a function can appear in a solution. This constraint is based on the assumption that tools only have a function at most once in their annotation. If a tool is introduced that uses a function more than once this constraint needs to be loosened and the amount of resulting solutions will increase. The amount of solutions is also limited with the GATA approach because the tool set used for testing is relatively small. Thus, there is not a lot of room for tools having overlapping GATA expressions. An overview of general differences can be found below in Table 7.

As can be seen in the overview, GATA-adapted APE is a more specialized version of APE. It allows users to express their intent on a higher level and receive a smaller more relevant set of solutions. However, it comes with the loss of generality of APE.

Property	APE	GATA-adapted APE
Requires workflow specification and domain knowledge	yes	yes
Workflow specification is defined	Within domain	Independent using GATA
Can be used for what kind of domains	Any	Only Geoscience related
Needs additional domain knowledge	No	Tools need a GATA annotation
Needs post-process step	No	Yes
Inherently generates a small and focused set of solutions	No	Yes

Table 7: An overview of the similarities and differences between APE and GATA-adapted APE

6 Conclusion

In this thesis I showed that the usage of a semantic language as an input for a workflow system is possible. I used four research questions as a basis for this thesis.

1. How can input semantically describe the goal?
2. How can this input be used in a workflow synthesis system?
3. How can it relate such input to concrete tools?
4. How can we translate the semantic input and domain knowledge into constraints for a SAT-based system?

GATA expresses the required calculations semantically which answers Research Question 1 (RQ1). I translated GATA expressions from the workflow specification and the additionally annotated tools from the domain knowledge into constraints that are usable by the synthesis algorithm of APE which answers RQ2. These constraints reduce the space of possible workflows that have the GATA specification as their representation. This was only possible as the tools also have a semantic representation in GATA which answers RQ3. However, not all solutions in the result represent the GATA expression of the workflow specification. A post-process step where I recreated the original expression from a workflow made sure only correct workflows are returned. The whole process allows a user to express their intent in GATA and thus independent from the domain which answers RQ4.

I showed that using GATA as a workflow specification is able to achieve results. However, for solutions longer than a length of 9, the generations of the CNF formulas for the SAT solver can easily take over 10 seconds. The same holds for the post process verification. This process takes longer than a couple of seconds on graphs with many multiple parameter functions. This greatly impacts the usage when it is run many times in succession for making small edits to the workflow specification. The graph reconstruction algorithm I used is naive and creates all possible graphs before starting to compare them. Due time constraints, I was not able to improve on it further. There is a lot of room for improvement and it is worth it to try to explore this approach, especially with eye on full automation in QuAnGIS. The guarantee that the workflows comply with the input is required for a user to be able to trust the system.

6.1 Relevance in AI

In general there is a gap between natural language and machine code. We try to close this gap from either side. From the machine code, we abstract and create programming languages and from the natural language

side we try to automatically generate programs from descriptions. To automatically generate workflow from a natural language description we need to close the gap from natural language to the logical constraints for a synthesis algorithm.

In this thesis, I closed the gap from a little by automating the translation from GATA to logical constraints for the workflow synthesis API, APE. GATA serves as an algebra that captures the calculation necessary to solve a Geoscience problem. Thus this thesis contributed to the automation of generating workflows from a Geoscience problem.

6.2 Future Work

The translation from SLTL to CNF can take upwards of 10 seconds for certain solution lengths. A big part of this is the translation from the propositional formula to the CNF. The prototype used a specific library to handle the propositional formulas and the transformation to CNF. There is no guarantee that it translates in the most efficient way possible. There might be a method that is more efficient than this library.

Another way to reduce transformation time is to research the impact of the SLTL formulas used. I defined the set of constraints before experimenting with the post-process step. The removal of certain constraints can reduce the problem set up time with minimal increases in amount of incorrect solutions. Graph filtering can offset this change and ensure correctness.

The graph filter algorithm can also be improved. It uses a naive approach of creating all possible graph combinations. It only compares at the end when all graphs have been created. This can be improved by partial comparing. After combining the expressions of two graphs already compare the partial graph to the original input. This can greatly reduce the amount of generated graphs in exchange for comparing more often.

Another way to improve the graph filter algorithm is to take into account the data types of GATA. GATA expressions are typed and so its inputs. Thus, not all expressions can fit on all the data leaves of another expression. This check will take barely any time and will reduce the amount of graphs to compare. But the algorithm needs to be made aware of the types in GATA. I had not enough time to develop it, but it is worth to investigate this in the future.

This thesis focused on using GATA as constraints for ArcGIS. It would be worth while to annotate tools in a different domain with GATA and compare this to the results of ArcGIS tools. One of the goals of GATA is using it as a single input for workflows in multiple domains. This thesis has shown it works for ArcGIS tools. Investigation is necessary to show that GATA can be used this way.

Another improvement is the test set. The set of tools and examples used to test are relatively small compared to the actual tool set available in ArcGIS. It is enough to prove it is worth investigating further, but the tool set is not representative for actual usage.

Finally, it would be interesting to test if geoscientists would like to use GATA as workflow specification. User studies should make clear if the language is easy to use or if it is just use full as part of QuAnGIS.

References

[1] APE Imagemagick use case example. `https://github.com/sanctuuary/APE_UseCases/tree/master/ImageMagick`. accessed: 10.08.2020.

[2] D. W. Allen. *Getting to know ArcGIS modelBuilder*. Esri Press, 2011.

[3] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pages 423–424. IEEE, 2004.

[4] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, Y. Zhu, et al. Bounded model checking. *Advances in computers*, 58(11):117–148, 2003.

[5] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems (TODS)*, 4(4):397–434, 1979.

[6] A. Desai, S. Gulwani, V. Hingorani, N. Jain, A. Karkare, M. Marron, and S. Roy. Program synthesis using natural language. In *Proceedings of the 38th International Conference on Software Engineering*, pages 345–356, 2016.

[7] N. Eén and N. Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[8] E. A. Emerson. Temporal and modal logic. In *Formal Models and Semantics*, pages 995–1072. Elsevier, 1990.

[9] S. Gulwani. Dimensions in program synthesis. In *Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming*, pages 13–24, 2010.

[10] S. Gulwani, O. Polozov, and R. Singh. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119, 2017.

[11] V. Kasalica and A.-L. Lamprecht. Workflow discovery through semantic constraints: A geovisualization case study. In *International Conference on Computational Science and Its Applications*, pages 473–488. Springer, 2019.

[12] W. Kuhn. Core concepts of spatial information for transdisciplinary research. *International Journal of Geographical Information Science*, 26(12):2267–2276, 2012.

[13] A.-L. Lamprecht, S. Naujokat, T. Margaria, and B. Steffen. Synthesis-based loose programming. In *2010 Seventh International Conference on the Quality of Information and Communications Technology*, pages 262–267. IEEE, 2010.

[14] S. Naujokat, A.-L. Lamprecht, and B. Steffen. Loose programming with prophets. In J. de Lara and A. Zisman, editors, *Fundamental Approaches to Software Engineering*, pages 94–98, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[15] M. Palmblad, A.-L. Lamprecht, J. Ison, and V. Schwämmle. Automated workflow composition in mass spectrometry-based proteomics. *Bioinformatics*, 35(4):656–664, 2019.

[16] M. Pesic, H. Schonenberg, and W. M. Van der Aalst. Declare: Full support for loosely-structured processes. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 287–287. IEEE, 2007.

[17] M. Raza, S. Gulwani, and N. Milic-Frayling. Compositional program synthesis from natural language and examples. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[18] S. Scheider and H. Kruiger. Geo-analytical transformation algebra. *To be determined*, 2020.

[19] S. Scheider, R. Meerlo, V. Kasalica, and A.-L. Lamprecht. Ontology of core concept data types for answering geo-analytical questions. *Journal of Spatial Information Science*, 2020(20):167–201, 2020.

[20] S. Scheider, E. Nyamsuren, H. Kruiger, and H. Xu. Geo-analytical question-answering with gis. *International Journal of Digital Earth*, pages 1–14, 2020.

[21] A. Solar-Lezama and R. Bodik. *Program synthesis by sketching*. Citeseer, 2008.

[22] B. Steffen, T. Margaria, and B. Freitag. Module configuration by minimal model construction, 1993.

[23] R. J. Trudeau. *Introduction to graph theory*. Courier Corporation, 2013.

[24] M. Veanes and A. Saabas. On bounded reachability of programs with set comprehensions. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 305–317. Springer, 2008.

[25] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, et al. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic acids research*, 41(W1):W557–W561, 2013.

[26] H. Zhang, E. Horvitz, and D. C. Parkes. Automated workflow synthesis. 2013.