CYBERPATTERNS: Linking Data Analytics To Reusable Knowledge

Hong Zhu, *Senior Member, IEEE,* Ian Bayley

***Abstract*—One of the most compelling challenges for data analytics is to obtain reusable, verifiable and transferable knowledge from data. One solution to this is the pattern-oriented approach to knowledge representation proposed by this paper. The foundation of the approach is a formal theory of patterns, including a formal language for defining them, and an algebra of operations for composing patterns and instantiating them. This paper outlines a roadmap for the study of so-called cyberpatterns: the patterns of cyberspace. It explores the scope of research, views the current state of the art and identifies the key research questions.**

I. Introduction

Cyberspace is making available a vast and ever increasing amount of data about the physical world, manmade systems, human behaviours and much else. This is thanks to the rapid advances in cloud computing, the Internet and wireless communication, smart devices, mobile computing, and Internet of Things technology. Consequently, data analytics has boomed into an active scientific / technical interdisciplinary research area. One compelling challenge for researchers in this field is to obtain, from data, knowledge that is reusable, testable, verifiable and transferable, so that learning and reasoning can be effectively integrated [1].

Much of that extraction depends on machine learning techniques, however, and these have many shortcomings. The results of these techniques do not come with the human-comprehensible explanations needed to validate, verify and test them. They may be short-lived too because they apply only to the currently available data set. Thus, they are not as reusable for different users in different situations. Finally, the results are often in a tool-specific format that is not transferable to other applications.

To overcome these problems, statistical learning and symbolic reasoning need to be combined in an effective manner, perhaps with knowledge representation playing a crucial role as Shoham argued [2]. However, in the past three decades, these two fields have mostly been developed separately by distinct research communities. Those working on the integration of symbolic and connectionist paradigms of AI are among the few exceptions; see [3] for a recent survey.

The solution proposed by this paper is a pattern-oriented approach that aims at bridging the gap between data based machine learning and reusable knowledge. It consists of two parts:

- *A formal theory of pattern*, which consists of a formalism of knowledge representation in the form of patterns, a set of operators that allow patterns to be composed and instantiated, and a collection of algebraic laws on these operators to enable reasoning about and processing of knowledge. The theory of patterns is generalised from its original use in software design to any subject domain, including the new area of cyberspace.
- *A pattern-oriented research methodology* that promotes study of a subject domain by systematically addressing a set of interrelated research questions with focus on patterns. As Cao pointed out recently [4], to achieve full the potential of data science, a discipline-wide effort and corresponding methodology is required.

II. Pattern-Oriented Research Methodology

In general, a pattern represents a discernible regularity in nature, manmade systems, human behaviours, etc. It can be seen as either a template from which instances can be created (the prescriptive view) or an account of recurring observable phenomena (the descriptive view). The latter makes it possible to predict regularities in a subject domain and is analogous to a scientific theory. When the subject domain is as complex as cyberspace, there may be a large number of interacting patterns that each describe and predict a subset of recurring phenomena. These patterns are all interrelated and they can be composed with each other.

The research questions for a pattern-oriented research methodology include devising ways to do the following:

- *identifying the patterns* manually, semi-automatically or even fully automatically e.g. from data mining and machine learning techniques;
- *documenting patterns* in a human comprehensible form with context information such as its applicability conditions, known problems, related other patterns, etc.;
- *specifying patterns* formally in a machine-readable unambiguous form using an appropriate formalism;
- *validating patterns* to ensure they correspond to what was intended;
- *naming patterns* to enlarge the vocabulary of discourse, so that patterns as fragments of a domain knowledge can be gradually accumulated and integrated into a theory;
- *classifying or categorising patterns* to clarify their relationships and to enable human understanding of them;
- *investigating the interactions between patterns* so that complicated phenomena can be recognised and non-trivial applications can be delivered;
- *devising mechanisms* for detecting pattern occurrences and predicting their occurrences in a dynamic system and evolving world;
- *facilitating the instantiation of patterns* when an instance of a particular pattern is required.

The pattern oriented research methodology attempts to answer all these questions systematically in a disciplined way that aims to link data analysis and machine learning to reusable, testable, verifiable and transferable knowledge.

This methodology would build on more than 20 years of research on software design patterns; see [5] for the most cited work. That research has led to a pattern-oriented software design method, which is now a common practice, in spite of the elusive nature of software design. In general, patterns are self-contained encapsulations of domain knowledge of a complex subject domain. A methodology based on patterns, in view of previous experience with software design, would therefore likely have the following benefits.

- Patterns act as highly comprehensible and learnable documentation for human users.
- Patterns can relatively easily be tested, validated and formally specified independently of other patterns.
- Pattern applicability can be recognised easily.
- Patterns can be flexibly combined, in a manner which can be formally defined and performed by applying operators that obey algebraic laws.
- Pattern instantiation and detection can be assisted by automated tools, as can reasoning about the relationship between patterns.

In addition, interactions between patterns are another important part of domain knowledge, which can also be formally defined. Patterns make it possible to develop incrementally a complete picture of the subject domain in the form of ever-expanding pattern catalogues, which can still be useful even when incomplete.

III. Patterns in Cyberspace

Cyberpatterns are the patterns found within cyberspace. As they are so ubiquitous, this prompts the application of the pattern-oriented research methodology to the study of cyberspace. Here, we discuss both the scope of the research and the core research questions to be addressed. This gives a road map for future research.

As discussed previously, cyberspace is a complex subject domain with many phenomena that can be observed, recorded and detected as patterns. Here are some areas in which patterns have already been studied by researchers in various research communities. Patterns can be found in

- *User behaviour* in accessing various internet applications, so that abnormal behaviour can be spotted as part of intruder detection.
- *Environmental data* as collected through sensors and smart devices, in order to improve energy efficiency, operation effectiveness and other qualities associated with the Internet-of-Things.
- *Normal workload* of a cluster of computers, server or internet service, in order to improve system performance and efficiency through load balancing and power management.
- *Abnormal workload* variations to detect system failures and malicious attacks on the system.
- *Network traffic*, since noticing that the activities of some users are time and date-dependent makes optimisation possible.
- *Social networks*, such as the Small World and Power Law of scale-free networks and the propagation of "fake news".

Other kinds of patterns include:

- *Attack patterns*, the regularities observed in the dynamic characteristics of the techniques used by hackers to attack a resource on the internet.
- *Security design patterns*, like software design patterns but applied to the mechanisms used to prevent, detect and mitigate security attacks. [6], [7]
- *Vulnerability patterns*, or patterns in the vulnerabilities of these security mechanisms, though they are not usually described as patterns in the literature.
- *Digital forensic patterns*, the guidelines for investigating various types of digital computer crime.

Much research effort has gone into identifying patterns in individual subject domains. Existing techniques for this include knowledge engineering, statistical data analysis, data mining, machine learning, etc. However, from the point of view of a pattern-oriented research methodology, much more work is needed to understand how patterns interrelate and interact not only within a domain but also more importantly across domains.

Relationships between patterns in a domain include one pattern being the subpattern of another or one pattern being, through composition, a part of another more complex pattern. Relationships across domains between patterns of different types are of great importance to the understanding of cyberspace and to the effective protection of Internet resources and infrastructure. Fig. 1 is a first attempt at depicting some of these relationships.

There are two key research questions for a pattern-oriented research methodology that aims at developing theories of a subject domain as human knowledge rather than just machine-

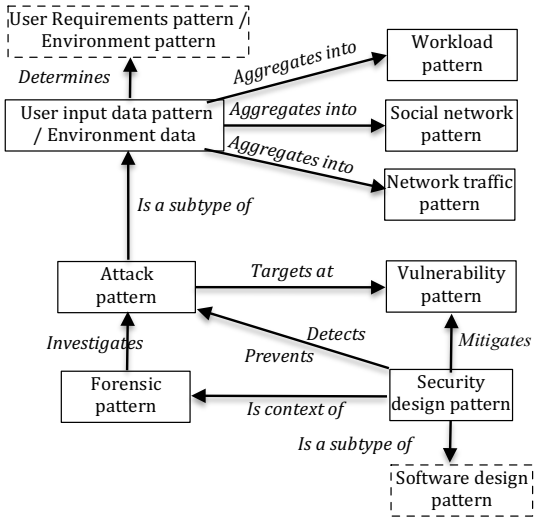

Fig. 1. Potential relationships between different types of cyberpatterns.

readable data. One is how to describe, document and specify patterns in a comprehensible, testable, verifiable and reusable form? However, since we also require machine processing of the knowledge captured by patterns, a second question is how can cyberpatterns be stored, retrieved and applied automatically with the support of computer software.

To lay a solid foundation for answering these questions, in the next two sections, we generalise the formal theory of patterns beyond just software design and demonstrate that it can be applied to a domain completely unconnected with software design.

IV. Pattern As A Knowledge Representation Formalism

A pattern in a specific subject domain consists of a collection of elements of various types. Each type typically represents a kind of artefact in a subject domain, or an atomic event. Each element can be characterised by a number of attributes or features and the elements are connected by relations.

A. Subject Domain

A subject domain can, accordingly, be characterised as a collection of all possible element types and relationships that may hold between them. The elements and relationships may have some properties universally satisfied by all the elements and relations. These are called the axioms of the domain.

Definition 1: (Domain) A *subject domain* (or simply *domain*) $\mathscr{D}$ is a triple $\langle \mathbf{T}, \mathbf{R}, \mathbf{Ax} \rangle$, where

- $\mathbf{T}$ is a finite collection of element types. For each type $T \in \mathbf{T}$, there is a finite set A_T of attributes associated with the elements of T. For each attribute $a \in A_T$, the value of a for element e is written as $e.a$, and this has the (data) type D_a.
- $\mathbf{R}$ is a finite collection of relations on $\mathbf{T}$. For each $R \in \mathbf{R}$, R is a k-ary relation on $T_1 \times \cdots \times T_k$, where $T_i \in T, i = 1, \cdots, k$. The proposition that a collection of elements $e_1, e_2, \cdots, e_k$ of types $T_1, T_2, \cdots, T_k$ respectively satisfies relation R is written as $R(e_1, e_2, \cdots, e_k)$.
- $\mathbf{Ax}$ is a finite collection of axioms about elements and their relations. Each axiom $Ax \in \mathbf{Ax}$ is a well-formed logic formula that is constructed from expressions of the form of $e.a$ and $R(e_1, e_2, \cdots, e_k)$, as described above, but also operations and relations on the attribute data types (e.g. $+, -, <, >$), equality (i.e. $=$ and $\neq$), set constructions in the form of $\{x_1, \cdots, x_n\}$ or $\{x | p(x)\}$, set operators (i.e. $\cap, \cup, -$) and set relations (i.e. $\in, \subset, \subseteq, \supset, \supseteq$), logic connectors *and* ($\wedge$), *or* ($\vee$), *not* ($\neg$), *implication* ($\Rightarrow$), *equivalence* ($\Leftrightarrow$), and quantifiers *for-all* ($\forall$) and *exists* ($\exists$). $\square$

For the sake of readability, in this paper we use floor tile layout patterns to explain and illustrate our notions and notation, rather than software design patterns or cyberpatterns, both of which are less comprehensible.

Example 1: (Domain of Floor Tile Layout)

Consider the layout of floor tiles as the subject domain. The elements are all of one type Tile with the following attributes:

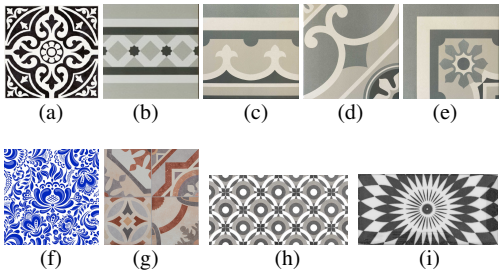

Fig. 2. Tiles of different attributes.

- $Shape : \{Square, Rectangle\}$, indicating whether the tile is in the shape of a square or a rectangle.
- $Width, Length : Integer$, whose value is the width (or length) of the tile in mm.
- $Image : JPEG$, whose value is the image of the tile in JPEG format;
- $CSym, HSym, VSym, DSym : Boolean$, which indicates whether the image is centrally, horizontally, vertically or diagonally symmetric, with centrally symmetric meaning that every line through the centre is symmetric about that centre;
- $HCon, VCon : Boolean$, which indicates whether or not two tiles of that image can be laid out next to each other horizontally (or vertically). We then say the image of the tile is horizontally (or vertically) connectable.

Fig. 2 shows a number of tiles with different attributes. Tiles (a)-(g) are in the shape of square, while (h) and (i) are rectangles. Tiles (a) and (h) have centrally symmetric images; (b) is both horizontally and vertically symmetric, but not centrally symmetric; (c) is vertically symmetric, but not horizontally symmetric. Tiles (d) and (e) are diagonally symmetric; (f) and (g) are not symmetric at all. Tiles (a), (b), (f), (h) and (i) are connectable both horizontally and vertically, but (c) is only horizontally connectable, while tiles (d), (e) and (g) are not connectable at all.

The relations on tiles important for layout are as follows:

- $OnLeft : Tile \times Tile$, where $OnLeft(x, y)$ means that tile x is placed on the left of y.
- $Below : Tile \times Tile$, where $Below(x, y)$ means that tile x is placed below y.
- $Rot90, Rot180, Rot270 : Tile$, where $Rot90(x)$ means that tile x is laid with the image rotated 90^o clockwise, and $Rot180$ and $Rot270$ are similar.

Here are some example axioms about tiles.

$$\forall x \in Tile.(x.Shape = Square \Leftrightarrow x.Width = x.Length)$$
$$\forall x \in Tile \cdot (x.Shape = Square \wedge x.CSym) \Rightarrow (x.HSym \wedge x.VSym)$$
$$\forall x \in Tile.(x.VSym \Rightarrow x.HCon)$$
$$\forall x \in Tile.(x.HSym \Rightarrow x.VCon)$$

These axioms state, in order, that square tiles have equal width and length, that centrally symmetric square tiles are also vertically and horizontally symmetric, that vertically symmetric tiles are horizontally connectable and that horizontally symmetric tiles are vertically connectable. $\square$

Axioms like these represent knowledge that is universally applicable in the subject domain. Element types, attributes and relations also represent knowledge that is often found in the form of ontology and which can often be obtained from data through feature extraction and data mining. Image processing could be used to deduce symmetry and connectability of tiles. The pattern oriented research methodology we promote in this paper uses all of these techniques alongside human knowledge engineering to create reusable, validatable, testable, comprehensible and transferable knowledge.

B. Instances of Phenomena

In a given subject domain, it is possible to observe certain phenomena which consists of a number of elements, i.e. artefacts and events of the domain, related to each other in a particular way. To recognise a phenomenon as an instance of a pattern is to recognise its constituent elements, noting their types and the values of their attributes, and noting also how the elements relate to each other.

Definition 2: (Instance) Given a subject domain $\mathscr{D} = \langle \mathbf{T}, \mathbf{R}, \mathbf{Ax} \rangle$, an instance ϕ is an ordered pair $\langle E, R \rangle$, where

- $E = \{e_1, \cdots, e_n\}$ is a collection of particular elements $e_i, i = 1, \cdots, k$, of type $T_i \in T$ in the domain $\mathscr{D}$, and
- $R = \{r_1, \cdots, r_m\}$ is a set of relations that hold between elements in E. □

Example 2: (An Instance of Floor Layout)

For example, Fig. 3 shows an instance of floor tile layouts. In this instance, elements are $E = \{e_{i,j} | i, j = 0, \cdots, 3\}$, where $e_{i,j}$ is the tile laid in row i column j. They have the following attributes:

$$\begin{aligned}
&\forall x \in \{e_{0,0}, e_{0,3}, e_{3,0}, e_{3,3}\} \cdot (x.Image = Corner.jpeg) \\
&\forall x = \{e_{0,1}, e_{0,2}, e_{1,0}, e_{1,3}, e_{2,0}, e_{2,3}, e_{3,1}, e_{3,2}\} \\
&\quad \cdot (x.Image = Border.jpeg) \\
&\forall x \in \{e_{1,1}, e_{1,2}, e_{2,1}, e_{2,2}\} \cdot (x.Image = Center.jpeg)
\end{aligned}$$

The following rules determine the other attributes:

$$\begin{aligned}
&\forall x \in Tile.(x.Image = Corner.jpeg \Rightarrow x.DSym) \\
&\forall x \in Tile.(x.Image = Border.jpeg \Rightarrow x.VSym) \\
&\forall x \in Tile.(x.Image = Center.jpeg \Rightarrow x.DSym) \\
&\forall x \in Ti.(x.Shape = Square)
\end{aligned}$$

The central part is laid out with the tiles rotated, i.e.

$$Rot90(e_{1,2}), Rot180(e_{2,2}), Rot270(e_{2,1})$$

The borders and corner tiles are also rotated as follows, relative to the three tiles on top $e_{0,0}, e_{0,1}, e_{0,2}$.

$$\begin{aligned}
&Rot90(x), for\ x = e_{0,3}, e_{1,3}, e_{2,3} \\
&Rot180(x), for\ x = e_{3,1}, e_{3,2}, e_{3,3} \\
&Rot270(x), for\ x = e_{1,0}, e_{2,0}, e_{3,0}
\end{aligned}$$

The relationships between the elements are as follows.

$$\begin{aligned}
&OnLeft(e_{i,j}, e_{i,j+1}), for\ i = 0, \cdots, 3, j = 0, \cdots, 2. \\
&Below(e_{i+1,j}, e_{i,j}), for\ i = 0, \cdots, 2, j = 0, \cdots, 3. \quad \square
\end{aligned}$$

Fig. 3. An Instance of Tile Laid Floor.

C. Patterns

Definition 3: (Pattern) A *pattern* Π in a domain $\mathscr{D} = \langle \mathbf{T}, \mathbf{R}, \mathbf{Ax} \rangle$ is an ordered pair $\langle V, P \rangle$, where

- $V = \{v_1 : TE_1, \cdots, v_k : TE_k\}$ is a collection of elements v_i of type TE_i, which is a type expression. V defines what are the elements in the pattern.
- P is well-formed logic formula (i.e. a predicate) with $\{v_1 : TE_1, \cdots, v_k : TE_k\}$ as its free variables. It specifies the constraints on the elements and the relationships between the elements in the pattern.

The semantics of pattern $\Pi = \langle V, P \rangle$, written $Sem(\Pi)$, is that

$$\exists v_1 \in TE_1, \cdots, \exists v_k \in TE_k \cdot P.$$

A type expression TE can be a type $T \in \mathbf{T}$, or a power set type $\mathbb{P}(TE)$ if TE is a type expression. Here, the values of a power set type $\mathbb{P}(T)$ are non-empty finite sets of elements of type T. □

Example 3: (An Example of Pattern: Bordered Centre)

For example, a common pattern of floor tile layout is the so-called "bordered-centre" (BC), in which the centre is an $n \times m$ matrix of tiles surrounded by matching borders and corner tiles. Fig. 3 is an instance of such a pattern. We can formally specify it by defining the set of variables with $Var(BC) \triangleq \{X : \mathbb{P}(Tile)\}$ and $Pred(BC)$ as follows.

The layout is an $(n+2) \times (m+2)$ matrix (with $n, m > 0$) in which the tiles are laid one next to another:

$$X = \{x_{i,j} | i = 0, \cdots, n+1, j = 0, \cdots, m+1\}.$$

Informally, $x_{i,j}$ is the tile laid on row i column j.

$$\begin{aligned}
&OnLeft(x_{i,j}, x_{i,j+1}), for\ i = 0, \cdots, n+1, j = 0, \cdots, m; \\
&Below(x_{i+1,j}, x_{i,j}), for\ i = 0, \cdots, n, j = 0, \cdots, m+1;
\end{aligned}$$

The properties of the central tiles are:

$$x_{i,j}.Image = x_{1,1}.Image, for\ i = 1, \cdots, n, j = 1, \cdots m.$$

The properties of the non-corner border tiles are:

$$\begin{aligned}
&x_{0,1}.VCon \wedge (x_{0,1}.Width = x_{1,1}.Width) \\
&\wedge \forall j \in \{1, \cdots, m\} \cdot (x_{0,j}.Image = x_{0,1}.Image \\
&\quad \wedge x_{n+1,j}.Image = x_{0,1}.Image \wedge Rot180(x_{n+1,j})) \\
&\wedge \forall i \in \{1, \cdots, n\} \cdot (x_{i,0}.Image = x_{0,1}.Image \wedge Rot270(x_{i,0}) \\
&\quad \wedge Rot90(x_{i,m+1}) \wedge x_{i,m+1}.Image = x_{0,1}.Image)
\end{aligned}$$

The properties of the corner tiles are:

$$x_{0,0}.Image = x_{0,m+1}.Image,$$
$$= x_{n+1,0}.Image = x_{n+1,m+1}.Image,$$
$$x_{0,0}.Width = x_{0,0}.length = x_{1,1}.Width,$$
$$Rot90(x_{0,m+1}), Rot180(x_{n+1,m+1}), Rot270(x_{n+1,0}). \quad \square$$

It is easy to see that the tile layout given in Fig. 3 is an instance of pattern BC because the conditions of pattern BC are all true when we assign element $e_{i,j}$ in the instance to variables $x_{i,j}$ in the pattern.

Definition 4: (Satisfaction) Let $\mathscr{D}$ be a given domain, $\phi = \langle E, R\rangle$ be a given phenomenon and $\Pi = \langle V, P\rangle$ be a pattern in domain $\mathscr{D}$. If there is a type preserving assignment α of the variables in V to elements in E, such that $[\![P]\!]_{\alpha,R} = true$, we say that phenomenon ϕ is an instance of pattern Π, and write $\phi \vDash \Pi$. $\square$

Note that, $[\![P]\!]_{\alpha,R} = true$ means that the evaluation of P under assignment α with the conditions of R is true. The detailed definition is omitted for the sake of space.

The above definition of the notion of pattern and satisfaction enable us to deal with the recognition of an instance of a pattern as the evaluation of logic formulas in a finite structure. For example, to show that the floor layout in Fig. 3 is an instance of the BC pattern, we simply need to assign variables in the pattern to particular elements in the instance and thereby prove all the conditions of the pattern are true.

V. Reasoning About And Applying of Patterns

Now, let's see how to reason about patterns, and to apply patterns.

A. Subpattern Relation

Many kinds of relationships between patterns can be defined in predicate logic so that the verification of such a relationship can be converted into logic inference problems. A typical example of such relationships is *sub-pattern*.

Definition 5: (Sub-Pattern and Equivalent Patterns)

Let Π and Ψ be two patterns in a domain $\mathscr{D}$. We say that Π and Ψ are *equivalent*, and write $\Pi \approx \Psi$, iff $Sem(\Pi) \Leftrightarrow Sem(\Psi)$. We say that Π is a *sub-pattern* of Ψ, and write $\Pi \preccurlyeq \Psi$ iff $Sem(\Pi) \Rightarrow Sem(\Psi)$. $\square$

For example, for the bordered-centre floor tile layout pattern BC, if we add a condition that the tiles in the central are horizontal and vertical connectable, we obtain a sub-pattern, which we called it BC-HVCon. If we replace the connectability conditions of BC-HVCon with the condition that the image of tiles in the central block is central symmetry, then the result pattern, called BC-CSym, is a sub-pattern of BC-HVCon. This is because we can prove from the axioms that central symmetry implies horizontal and vertical connectability.

B. Operators on Patterns for Composition and Instantiation

Operators on patterns have been shown to be particularly valuable in the context of software design patterns where they can be used to define pattern compositions and instantiations [8]. Now we revisit those definitions to demonstrate their applicability to a more general notion of patterns.

Definition 6: (Pattern Operators) Let Π and Θ be any given patterns of a domain $\mathscr{D}$, $V_\Pi = Var(\Pi) = \{x_i : T_i | i = 1, \cdots, n\}$ and $Pred(\Pi) = p(x_1, \cdots, x_n)$.

- *Restriction*. Let c be a predicate on V_Π. $\Pi[c]$ is the pattern that
$$Var(\Pi[c]) \triangleq Var(\Pi),\ and$$
$$Pred(\Pi[c]) \triangleq p \wedge c.$$
- *Superposition*. Assume that $V_\Pi \cap Var(\Theta) = \emptyset$. $\Pi * \Theta$ is the pattern such that
$$Var(\Pi * \Theta) \triangleq Var(\Pi) \cup Var(\Theta),\ and$$
$$Pred(\Pi * \Theta) \triangleq Pred(\Pi) \wedge Pred(\Theta).$$
- *Generalisation*. Assume that $x = x_i \in V_\Pi, X \notin V_\Pi$. $\Pi(x \Uparrow X)$ is the pattern such that
$$Var(\Pi(x \Uparrow X)) \triangleq V_\Pi - \{x_i : T_i\} \cup \{X : \mathbb{P}(T_i)\},$$
$$Pred(\Pi(x \Uparrow X)) \triangleq \forall x_i \in X.p(x_1, \cdots, x_i, \cdots, x_n).$$
- *Flattening*. Assume that $X = x_i \in V_\Pi$ and $T_i = \mathbb{P}(T)$. $\Pi(X \Downarrow x)$ is the pattern such that
$$Var(\Pi(X \Downarrow x)) \triangleq Var(\Pi) - \{x_i : \mathbb{P}(T)\} \cup \{x : T\},$$
$$Pred(\Pi(X \Downarrow x)) \triangleq p(x_1, \cdots, \{x\}, \cdots, x_n).$$
- *Lifting*. Assume that $X = \{x_i | i = 1, \cdots, m\} \subset V_\Pi$, $m \leq n$. $\Pi(X \uparrow Xs)$ is the pattern such that
$$Var(\Pi(X \uparrow Xs)) \triangleq \{xs_i : \mathbb{P}(T_i) | i = 1, \cdots, n\},$$
$$Pred(\Pi(X \uparrow Xs)) \triangleq \forall x_1 \in xs_1, \cdots, \forall x_m \in xs_m \cdot$$
$$\exists x_{m+1} \in xs_{m+1}, \cdots, \exists x_n \in xs_n \cdot p(x_1, \cdots, x_n). \quad \square$$

Note that, first, the operators on patterns are defined constructively. They can be easily implemented as syntactical transformations of the logical statements of the patterns. Second, a correct renaming of the variables in a pattern will not affect its meaning. In the sequel, we write $\Pi(x/x')$ to denote the systematic replacement in pattern Π of the variable x with variable x'.

We write pattern expressions with pattern variables, constants and operators to define predicates and functions on patterns. This results in a formalism of very powerful expressiveness to represent knowledge of a subject domain. The following example demonstrates how pattern operators can be used to represent domain knowledge, such as how to compose and instantiate patterns.

Example 4: (Application of Pattern Operators)

We start with a simple pattern H2 that consists of two tiles A and B of the same image that A is laid on the left of B.

$$Var(H2) \triangleq \{A, B : Tile\}$$
$$Pred(H2) \triangleq OnLeft(A, B) \wedge (A.Image = B.Image)$$
$$\wedge A.HCon$$

A horizontal line (HL) of tiles that laid one next to another can be defined by applying the lift operator with a constraint condition as follows.

$$HL \triangleq H2(A \uparrow As \triangleq \{A_i | i = 1, \cdots, n\})$$
$$[Bs \triangleq \{B_i | OnLeft(A_i, B_i), i = 1, \cdots, n\}$$
$$\wedge \forall i \in \{1, \cdots, n-1\}.(A_{i+1} = B_i)].$$

Matrix is a pattern that consists of $n \times m$ tiles. It can be defined as a number of horizontal lines placed one below another. Thus, we have that

$$Matrix \triangleq HL(l \Uparrow ls = \{l_1, ..., l_n\}) \\ [\forall i \in \{1, \cdots, n-1\} \cdot ((|l_i| = |l_{i+1}|) \wedge Below(l_{i+1}, l_i))]$$

Here, the binary predicate $Below(x, y)$ on horizontal lines of equal length is defined as follows. Let $n = |x.l| = |y.l|$.

$$Below(x, y) \triangleq \forall i \in \{1, \cdots, n\} \cdot Below(x.l_i, y.l_i).$$

Similarly, we can generalise other attributes and relations of tiles to that of horizontal lines L of n tiles, even a matrix M of $n \times m$ tiles. New functions and predicates can also be defined. For example, let $M.Cols$ and $M.Rows$ denote the numbers of columns and rows of a matrix M. The width of a matrix M can be defined as follows.

$$M.Width \triangleq x.Width \times M.Cols, where\ x \in M.b.$$

Note that, in the above we used the so-called "modifier-dot notation" $M.x$ to denote the variable x in an instance or pattern variable M so that naming conflicts can be avoided without renaming the variables.

A matrix of identical tiles that are connectable, called Styled Matrix (SM), can be defined by applying the restriction operator on the Matrix pattern.

$$SM \triangleq Matrix[\forall x \in b.((x.Image = b_{1,1}.Image) \\ \wedge x.VCon \wedge x.HCon)].$$

The pattern Bordered Centre with connectivity (BC-HVCon) can be defined as being composed of top, bottom, left and right borders and a styled central matrix plus four corners. The left and right borders are Vertical Lines (VL), which can be defined as a matrix of one column with the condition that the tiles are vertically connectable.

$$VL \triangleq Matrix[Cols = 1 \wedge \\ \forall x, y \in b \cdot (x.Image = y.Image \wedge x.VCon)].$$

The Corners pattern specifies the constraints on the tiles placed on four corners of a bordered central layout.

$$Var(Corners) \triangleq \{cn : P(Tile)\}, \\ Pred(Corners) \triangleq (cn = \{cul, cur, cll, clr\}) \\ \wedge \forall x \in cn \cdot ((x.Image = c_{ul}.Image) \wedge x.DSym) \\ \wedge Rot90(c_{ur}) \wedge Rot180(c_{lr}) \wedge Rot270(c_{ll}).$$

Now, we apply the pattern operators to define pattern BC. First, we rename the variables of the borders to avoid naming conflicts in the pattern composition.

$$Top \triangleq HL(l/bt); \quad Bottom \triangleq HL(l/bb); \\ Left \triangleq VL(l/bl); \quad Right \triangleq VL(l/br).$$

BC can now be defined as follows.

$$BC \triangleq (Top * Bottom * Left * Right * SM * Corners) \\ [Connection],$$

where $Connection$ consists of the following conditions.

$$OnLeft(Left, Central) \quad OnLeft(Corners.c_{ul}, Top) \\ OnLeft(Central, Right) \quad OnLeft(Top, Corners.c_{ur}) \\ Below(Central, Top) \quad OnLeft(Bottom, Corners.c_{lr}) \\ Below(Bottom, Central) \quad OnLeft(Corners.c_{ll}, Bottom) \quad \square$$

C. Algebra of Pattern Operators

The operators on patterns obey a set of algebraic laws, which can be proven from the definitions of the operators. Fig. 4 shows some of the complete set of laws in [8].

1. Laws on Sub-pattern Relation $\preccurlyeq$:

$$\Pi \preccurlyeq \Pi$$
$$(\Pi \preccurlyeq \Theta) \wedge (\Theta \preccurlyeq \Psi) \Rightarrow (\Pi \preccurlyeq \Psi)$$
$$\Pi \preccurlyeq \Theta \wedge \Theta \preccurlyeq \Pi \Rightarrow \Pi \approx \Theta$$
$$FALSE \quad \preccurlyeq \Pi \preccurlyeq \quad TRUE$$

2. Laws on Restriction $[c]$:

$$(c_1 \Rightarrow c_2) \Rightarrow \Pi[c_1] \preccurlyeq \Pi[c_2]$$
$$\Pi[c][c] \approx \Pi[c]$$
$$\Pi[c_1][c_2] \approx \Pi[c_2][c_1]$$
$$\Pi[c_1][c_2] \approx \Pi[c_1 \wedge c_2]$$
$$\Pi[true] \approx \Pi$$
$$\Pi[false] \approx FALSE$$

3. Laws on Superposition $*$:

$$(\Pi * \Theta) \preccurlyeq \Pi$$
$$\Theta \preccurlyeq \Pi \Rightarrow \Pi * \Theta \approx \Theta$$
$$\Pi * \Pi \approx \Pi$$
$$\Pi * TRUE \approx TRUE * \Pi \approx \Pi$$
$$\Pi * FALSE \approx FALSE * \Pi \ \approx \ FALSE$$
$$\Pi * \Theta \approx \Theta * \Pi$$
$$(\Pi * \Theta) * \Psi \approx \Pi * (\Theta * \Psi)$$

4. Laws on $\Downarrow$ and $\Uparrow$:

$$\Pi(X \Downarrow x)(Y \Downarrow y) \approx \Pi(Y \Downarrow y)(X \Downarrow x)$$
$$\Pi(x \Uparrow X)(y \Uparrow Y) \approx \Pi(y \Uparrow Y)(x \Uparrow X)$$

5. Laws connecting $*$ with others:

$$\Pi[c] * \Theta \approx (\Pi * \Theta)[c]$$
$$\Pi(x \Uparrow X) * \Theta \approx (\Pi * \Theta)(x \Uparrow X)$$
$$\Pi(X \Downarrow x) * \Theta \approx (\Pi * \Theta)(X \Downarrow x)$$
$$\Pi(X \uparrow Xs) * \Theta \ \approx \\ (\Pi * \Theta)(X \uparrow Xs)(\mathbb{P}(V_\Theta) \Downarrow V_\Theta)$$

6. Laws connecting $\Uparrow$, $\Downarrow$ and $\uparrow$:

$$\Pi(x \Uparrow X)(X \Downarrow x) \approx \Pi$$
$$\Pi(X \Downarrow x)(x \Uparrow X) \approx \Pi$$
$$\Pi(X \uparrow Xs)(Xs \Downarrow Var(\Pi)) \approx \Pi$$
$$\Pi(x \Uparrow X) \approx \Pi(X \uparrow Xs)(Xs - X^{\uparrow} \Downarrow X)$$

7. Laws connecting $[c]$ with $\Uparrow$, $\Downarrow$ and $\uparrow$:

$$\Pi(X \Downarrow x) \approx \Pi[\exists x \cdot (X = \{x\})]$$
$$\Pi(x \Uparrow X) \approx \Pi[\exists X \cdot (\forall x \in X \cdot p)]$$
$$\Pi(X \uparrow Xs) \approx \Pi[\exists Xs \cdot (p_{X \uparrow Xs})]$$
$$\Pi[c](x \Uparrow X) \approx \Pi(x \Uparrow X)[c_{x \Uparrow X}]$$
$$\Pi[c](X \uparrow Xs) \approx \Pi(X \uparrow Xs)[c_{X \uparrow Xs}]$$
$$\Pi[c](X \Downarrow x) \approx \Pi(X \Downarrow x)[c_{X \Downarrow x}]$$

Fig. 4. Laws of Pattern Operators

Using laws like this, we can simplify pattern expressions to prove that two patterns are equivalent or that one is a sub-pattern of the other. One example is that for all patterns Π that contain variable X of power set type $\mathbb{P}(T)$, we have that

$$\Pi(X \Downarrow x) = \Pi[|X| = 1].$$

Such reasoning about patterns has been proven to be very useful for software design [8], [9].

For example, we can define Vertical Line (VL) patterns as a lift of Vertical Two (V2) similar to the way that HL pattern is defined, and then to prove that this definition is equivalent to what we have defined VL in the previous subsection.

In general, any pattern expression can be transformed into a unique normal form using the complete set of algebraic laws given in [8]. Therefore, the equivalence between two pattern expressions can be deduced by logic inferences of the equivalence of their normal forms using the axioms of the subject domain with a logic inference engine like SPASS as shown in [8], [9].

VI. Conclusion

Nobel Laureate Ernest Rutherford once pointed out that *"all science is either physics or stamp collecting"*. By physics, he meant clean, succinct principles that apply to diverse phenomena. By stamp collecting, he meant the act of cataloguing and organising large sets of observations. Since patterns are reusable knowledge about recurring phenomena, we believe they form a bridge from "stamp collecting" to "physics". This is because pattern-oriented research methods not only recognise, catalogue and organise observations, but also discover regularities, interrelationships and interactions. Such knowledge is the mother of clean, succinct principles.

Rutherford also pointed out that *"scientists are not dependent on the ideas of a single man, but on the combined wisdom*

of thousands of men, all thinking of the same problem, and each doing his little bit to add to the great structure of knowledge which is gradually being erected". The pattern-oriented research methodology offers such a platform for thousands of computer scientists to contribute to the construction of knowledge about cyberspace, an activity we have only just started. The first step towards this will be a common pattern specification language that is suitable for all subject domains. This is what we are working on at the moment.

REFERENCES

[1] L. G. Valiant, "Knowledge Infusion: In Pursuit of Robustness in Artificial Intelligence", in Foundations of Software Technology and Theoretical Computer Science, 2008. Editors: R. Hariharan, M. Mukund, V. Vinay, pp. 415-422, 2008.

[2] Y. Shoham, "Why Knowledge Representation Matters", C. ACM, Vol. 59, No. 1, pp47-49, Jan 2016.

[3] A. dAvila Garcez, *et al.*, "Neural-Symbolic Learning and Reasoning: Contributions and Challenges", Workshop on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches at the 2015 AAAI Spring Symposium, pp18-21, Mar. 2015.

[4] L. Cao, "Data Science: Challenges and Directions", C. ACM, Vol. 60, No. 8, pp59-68, August 2017.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns-Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[6] C. Blackwell and H. Zhu, (eds.), CyberPatterns: Unifying Design Patterns with Security Patterns and Attack Patterns, Springer, 2014.

[7] C. Blackwell and H. Zhu, (eds.), Proceedings of the Third International Workshop on CyberPatterns: From Big Data to Reusable Knowledge, in Proceedings of IEEE 8th International Symposium on Service Oriented System Engineering, April 2014.

[8] H. Zhu, and I. Bayley, "An Algebra of Design Patterns", ACM Transactions on Software Engineering and Methodology, Vol. 22, No. 3, Article 23. July 2013.

[9] H. Zhu and I. Bayley, "On the Composibility of Design Patterns", IEEE Transactions on Software Engineering Vol. 41, No. 11, pp1138-1152. Nov. 2015.