A Framework for Universal Problem Resolution with Continuous Improvement

by

Robert Leithiser

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 3, 2014

Keywords: Autonomous Learning Systems; Continuous Improvement Frameworks; Relational State Sequence; Problem Solving; Machine Learning; Sequences;

Approved by

John A. Hamilton, Jr., Committee Chair, Alumni Professor of Computer Science and Software Engineering
Kai H. Chang, Department Chair, Computer Science and Software Engineering
David A. Umphress, Associate Professor of Computer Science and Software Engineering
Alvin Lim, Associate Professor of Computer Science and Software Engineering

Abstract

This dissertation outlines a universal problem resolution framework implementing reasoning and improvement capabilities exhibited by human beings. The work reviews literature concerning major differentiations of human learning compared to contemporary artificial intelligence (A/I) systems. Deductions from the body of knowledge in mathematics, computer science, and human learning quantify the human capability for continuous and recursive self-reflection as a central differentiator. Analysis of capabilities of computational systems establishes that structural differences between the human brain and computational devices do not preclude computational implementation of human self-reflective capability. The work establishes self-reflective capability as an enabler for a co-recursive and recursive paradigm operable with unlimited depth and breadth for the problem of optimizing problem solving itself.

Discovering the general algorithm for the Tower of Hanoi is the base case used to show how simulation outputs can transform to higher order pattern recognition problems. The Tower of Hanoi provides a model that is common for any problem in that it includes an initial state, a desired outcome state, and an allowed transition state (where states allow multiple combinations of sub-states). This model is representable in the system and thus enabled for solution discovery. Detailed examples wherein the system pursues problem search spaces in a manner enabling autonomous self-improvement as a natural result of encountering new types of problems validate the thesis. Cooperating agents analyze solution path determinations for problems including those concerning

their own optimization. This spawns state transition rules generalizable to higher layers of abstraction resulting in new knowledge enabling self-optimization.

Major outcomes include:

1) Proof by example that a continuous improvement universal problem resolution framework is constructible using currently available software and hardware,

2) Production of a prototype that meets the thesis for a continuous improvement system – non-domain specific, extensible without reprogramming of the core system, lacking need of subject matter expert intervention; and executing within polynomial time complexity,

3) Rationale that the framework coupled with technology advancements generate optimal solutions using fewer resources than possible without such a framework.

Acknowledgments

I am grateful for many people for their help on this dissertation including my committee and faculty from Auburn, friends, family, and professional associates. Among these are:

My advisor and committee chair Doctor Hamilton for his practical expertise in simulation and software architecture as well as never-ending support, knowledge in encouragement and direction;

Other committee members: Dr. Umphress, Dr. Chang, and Dr. Lim for the benefits received from expertise in areas of software design, quality assurance, and distributed systems, as well as their consistent support for the my studies;

Other Auburn faculty: Dr. Levant Yilmaz whose class provided me foundational knowledge for modelling of relational states; Dr. Cheryl Seals who encouraged me to think outside of the box for modelling multi-agent gaming problems; JoAnn Lauraitis for all her help in administrative areas;

Friends and associates: Drew Minkin who spent many hours helping me with conceptual understanding and practical applications of machine learning and predictive analytics as well as providing feedback on the overall concept and design; Jordan Martz for his encouragement, brain-storming and insights; Robert McGraw for his feedback on readability; My personal and business counselor John Bowers, and friends that share my faith who upheld me in prayer including James and David Frederick.

My wife Elaine who has supported me for nearly 30 years in not only this pursuit but in all my work and academic pursuits and helped in the organization of the literature review;

My daughter Brooke who tirelessly searched the body of literature to find anything of relevance and helped extensively with grammar and style proofing;

My son Blake, a premier software engineer in his own right who provided me feedback on many of the implementation ideas.

Finally, I thank God through Jesus Christ my Savior for the grace and strength to enable me to finish this work and the wonders of His creation including the marvel of human learning that is the example and the inspiration behind the dissertation concept.

Table of Contents

List of Tables

List of Figures

1. INTRODUCTION

1.1 Motivation

The motivation for this dissertation comes from a desire to advance computational processing in the area of problem solving from a domain-centric orientation to a general-purpose orientation [1], universal in nature with intrinsic support for automated continuous improvement. This motivation concerns not only the generic capability to model and represent problems within a system, but also the ability to generically discover and optimize solutions to computational problems. A key assertion of this dissertation is that the ability to resolve problems universally in terms of their optimal solutions will provide an automated improvement mechanism in the problem solving system itself and minimize the interaction requirements of domain-specific subject matter experts (SMEs). This dissertation substantiates the underlying assumption that problem solving itself is an NP-complete problem and therefore benefits from and provides benefits to other problems in the NP-class due to the effectiveness of approximation techniques and ever-increasing computational capabilities, even if the optimal solution never achieves polynomial time.

A second motivation is to model the remarkable self-reflective component at work early on in human development whereby the mind intrinsically reflects on each experience in order to evaluate: (1) the success of varying approaches used to solve a problem; and (2) the methods of evaluation useful for success [2]. In addition, present at youth is a form of creativity that involves utilizing a solution from a prior domain and applying it in new ways to a completely different domain seems [3]. This capability appears to be part of the predictive aspect of human learning

and behavior whereby humans are able to quickly generalize and correlate current events with past experiences, learn to effectively filter vast amounts of sensory inputs, and make predictions [4]. The results of these predictions then help guide the process for filtering input data, analyzing events and, ultimately spawn creativity to realize efficiency improvements. Similar benefits should be realizable in a software framework that leverages self-reflection effectively.

A third motivation comes from the desire to simulate the human learning processes within a computational framework [5]. Even children at very young ages possess the ability to differentiate themselves from the rest of the animal kingdom through mastery of complex language. Language learning remains a puzzling issue for the fields of both neuroscience and psychology [6]. This unique ability to evolve from virtually no language competency to highly sophisticated communication in the space of just a few short years surpasses the achievements by even the most adventurous artificial intelligence experiments. One hope is that a continuously improving problem solving system may lead to insights in the language-learning process.

1.2 Research Goal

These motivations inspire the goal for a universal problem solving system. However, not all problems are solvable, so the term “resolution” better supports the concept of searching for the outcome to a problem. In addition, the term “system” implies a self-contained product but the dynamic nature of problems indicate a need for extensibility and evolvability indicating a framework. Based on this, the dissertation targets a feasibility proof of a universal problem resolution framework (UPRF) as an outcome of prototyping the concept.

1.3 Research Questions

These motivations raise four research questions:

1) Is it possible to build a universal problem resolution framework that continuously improves and learns from its own execution including automatically cross-applying concepts learned in other domains in the most optimal method possible given the information available?
2) What is the design of such a universal problem resolution framework (UPRF)? What are the constraints and requirements for a design that fully supports generic representation of problems, generic pursuit of problem solutions and continuous improvement utilizing an overarching set of processing components **without** the need for modifications of the actual components for the solving system? The specific subordinate questions under this are:
    a. How can UPRF generically encode problems from any domain without the need for redesign of the problem representation process? What is the level of effort required to represent various types of problems?
    b. How can it utilize the generic representation of a problem to explore possible solutions?
    c. How can it support a continuous improvement paradigm?
3) Is UPRF practical for various types of problems?
    a. How would UPRF model, solve, and improve with various types of problems?
    b. Can it scale beyond the Tower of Hanoi base case to more advanced scenarios including scenarios representing real-world industry problems?
    c. What Subject-Matter-Expert (SME) interactions are required? What aspects of the framework help minimize these interactions?

d. Does the architecture support scaling to large scenarios and provide capability to leverage emerging technologies?

1.4 Methodology

The dissertation addresses the research questions through the following chapters:

- Chapter 2 – Literature Review: This chapter reviews the literature covering concepts that form the foundations and provide impetus for a UPRF. Problem representation, simulation, complexity, human learning theory, recursion, co-recursion, relational algebra, and reflectivity are among the topics reviewed. The section concludes with a review of the research and the implications to feasibility for the UPRF concept.
- Chapter 3 – System Framework: This chapter outlines the basic design of UPRF. The goal of this chapter is to lay the groundwork to support a processing infrastructure for problem representation. This chapter shows the feasibility for the framework to support the data and processes associated with problem representation, simulation, and learning. It also provides insights into how a UPRF can be constructed, prescribing a set of constructs for relational problem representation that support simulation and solution discovery including higher order problem transformation.
- Chapter 4 – Operational Proof: This chapter targets the first research question (regarding feasibility) from the perspective of proving that problems can be generically represented, simulated, and analyzed for solution patterns to generate higher order problems within the framework that learn how to solve the lower order problems. Whereas the system design chapter focuses on structure, this chapter focuses on operation. It postulates constructs, provides an informal semantic proof and concludes with a proof-by-example scenario

using the Tower of Hanoi. The goal of the chapter is to demonstrate how multiple simulative solving instances generate knowledge within a system for solving higher order pattern recognition to find a general solution.

- Chapter 5 – Practicality: This chapter targets the third research question concerning operation of the framework for various applications. It provides additional use cases beyond the Tower of Hanoi to show how UPRF extends to other types of problems. It explore more complex scenarios including:
    a. K-peg variations of the Tower of Hanoi illustrating the ability to scale from a lower-level problem domain to higher-level domains.
    b. Multi-agent/deterministic/exclusively competitive or collaborative including an example with Tic-Tac-Toe.
    c. Single-agent/non-deterministic/random influencer including an example involving stock market trading.
- Chapter 6 – Conclusion and Further Research: This chapter summarizes the findings, the significance of the work, and provides insights for the fourth research question involving the practicality of UPRF. It explores enabling technologies that can make the massive amount of data tracking practical. This includes high-speed storage technology, distributed problem solving, and high-performance computing.

1.5 Chapter Summary

This chapter introduced the motivation and research goals associated with the objective of this dissertation, a universal problem-solving framework (UPRF). The motivation derives from drawing parallels between the factors driving humankind's advancements over time and software

frameworks leveraging a universal, continually improving model. Research goals associated with answering questions of feasibility, constructability, and practicality arise from this motivation and form the basis for the work of the dissertation.

2. LITERATURE REVIEW

2.1 Maslow and Technology

The technological progress of human civilization focuses on the improvement of solutions to problems facing the human species. A key differentiator between humans and other species is the ability to identify non-optimal aspects of our existence and utilize nature in the creation of continually improving remedies to challenges that detract from the human experience [7]. These remedies span across all of the aspects of human life and encompass all of the needs associated with motivation of human behavior.

Maslow provides a hierarchy of needs for human motivation [8]. This hierarchy depicts a pyramid with basic physiological survival at the base of the hierarchy and self-actualization at the top. The respective intermediate levels moving up the pyramid are safety, love/belonging, and esteem. Through history, human civilization has evolved by continuously devising improving methods to meet these needs. Invention is the cornerstone of the human existence and significantly differentiates us from all other species. Throughout history, no other species has shown such significant capability for progression as humans. While all species evolve to adapt to changing conditions, rarely does a species achieve a higher level of existence using technology.

2.2 Technology and Computer Systems

Technology is often thought of as an advent of the 20th century, however the word "technology" comes from Greek *τεχνολογία (technología)*; from *τέχνη (téchnē)*, meaning "art,

skill, craft," and *-λογία (-logía)*, meaning "study of" [9]. Technology marks the beginning of human civilization. Technological progress ultimately involves enabling progression within a system from an initial state to a goal state where the initial state is less than desirable and the goal state is the desired improvement. At the bottom of the Maslow's hierarchy, the invention of the wheel (epitomizing the beginning of civilization) optimizes the scenario of reducing the resistance between an object and its surface to minimize the effort required to move the object. At the top of the pyramid, the printing press facilitates the populace to enter into satisfaction associated with literary works.

In the modern era of computer software, higher level programming provides the ability to create automated information management systems that help humans across all the layers of Maslow's hierarchy – from scheduling flights to tracking finances to social media interaction – and everything in between [10]. Algorithms, improvements, and innovations within hardware and software continue to improve the capabilities of computer systems. A unique aspect of computational devices is the orientation toward solving problems across all domains. Computer systems are no longer only informational devices providing an answer to some sort of computation problem, but are bidirectional in nature [11] exerting control over systems such as financial markets, power grids, medical aids, and weapon systems. Computer systems solve problems that directly affect everything from human survival itself to geo-political stability.

2.3 Universality of Computational Frameworks

Modern computer hardware and operating system software platforms are able to run applications encompassing multiple domains without any pre-knowledge of the specific domain [12]. The same computational device using the same operating system that can run an accounting

package can also run word processing software, engineering simulations and data mining software. Programming languages provide support for creating application software that runs the full gamut of computational requirements and represents a universality class in the overall computer system framework.

Relational database management systems [13] using Structured Query Language (SQL) [14] provide a generic capability to represent a wide variety of data. Any information that can be understood in terms of values related to other values, including those related through functions, can be represented in a relational schema [15]. Relational databases have evolved to support object-oriented storage, another class of universality in the area of information representation.

More recently, data mining [16] software supporting business intelligence [17] and predictive analytics [18] such as MicroStrategy™ and 'R' have emerged with increasingly considerable capabilities that are further enhanced by hardware developments such as solid-state storage, parallel processing and by evolution of distributed computing networks. Similar to programming languages and operating systems, data mining has become increasingly generic [19] with the same data mining software supporting analysis of different domains.

2.4 The Universality Gap Pertaining to Problem Solving

Despite the universality of operating systems, programming languages, databases, and data mining software, domain-specific limitations persist when endeavoring to discover solutions to computational problems [20]. For example, data mining software may provide a generic platform for analyzing pattern data, but, for many scenarios, subject matter expertise is heavily needed in not only the problem definition aspect, but also in evaluating results and formulating actions from those results [21].

It is natural that problem solving tends to be domain-specific since problems that we encounter are by their very nature varied across the entire sphere of existence. For example, the solution for calculating a flight trajectory is vastly different from the solution for determining customer preferences based on prior purchases. Another example of domain-specificity is the Deep Blue system for playing chess [22]. While this system was able to defeat the best grandmaster in the world, it was not able to play any other games – even a game as simple as tic-tac-toe – without extensive re-programming [23].

The domain specificity not only concerns algebra, and correlative analysis, but also entails the algorithmic aspect for determining optimal solutions. Typically software designed to find the optimal solution for a problem focuses on the use of a specific algorithmic approach or a combination of approaches whether it be neural networks [24] dynamic programming [25], heuristic-based [26], genetic algorithms [27], simulation-based [28] approximation-oriented [29**]** or some other method. Without a doubt, certain types of problems benefit more from specific algorithmic approaches. Therefore, feedback mechanisms commonly found in problem optimization research are typically limited to merely evaluating the success of the algorithm and parameters [30] rather than a holistic approach that encompasses any algorithm and endeavors to determine the most optimal set of algorithms or the most optimal patterns for applying the algorithms.

2.5 The Case for Universal Problem Solving

While it is true that using even the highest-level programming languages (including those oriented towards problem solving such as PROLOG [31] and Python [32]) will yield different approaches to achieve optimality, this dissertation asserts there is a central issue that exists in

problem solving itself that accommodates universality. The evidence for this comes from the large family of NP-complete problems. An NP-complete problem is a problem that does not have an exact solution in polynomial time and is thus NP-hard in execution, but is in the class of P regarding verification since NP-Complete problem solutions are verifiable within polynomial time [33].

All NP-complete problems – some of which seem very different on the surface such as the zero subset problem, the traveling salesman problem (TSP) [34] and theorem proving are fundamentally reducible to each other through mathematical transformations. It has been shown that if one can find a polynomial-time solution to one NP-complete problem, then all NP problems would be solvable and thus that P=NP would be proven true [35]. The discovery that P=NP has incredible repercussions and, in fact, most mathematicians and computer scientists do not believe that P=NP is even possible. However, P!=NP also persists as an open question.

The ability to transform problems into a generic format is not limited to variations of NP-complete problems. Relational databases provide another example of the extensibility for modeling a problem within a common framework under the umbrella schema of the relational model, particularly through application of associative data modelling [36] and key/value pair approaches such as “noSQL” [37] that support generic representations. The best evidence for generic problem representation comes from the representation of knowledge in the human mind. Humans are able to link memories together without limitation even where the schematic attributes of the knowledge is completely different. Images, emotions, facts, and events are all seamlessly related together suggesting the concept of generic schema in the infrastructure of the human mind that is able to represent all information such that any piece of datum can be related to another [38].

Cognitive machines that utilize varying technologies with the ability to learn new information and improve themselves are yet another example supporting the concept of universal problem resolution. Such machines function in their environments for a particular purpose but integrate with the environment and receive feedback to improve in their operations [39]. Cognitive machines are often used for surveillance and sensing of events in the environment and share the following similarities:

- They have embedded (i.e., software-defined) signal processing for flexibility.
- They perform learning in real-time through continuous interactions with the outside environment.
- They utilize closed-loop feedback.

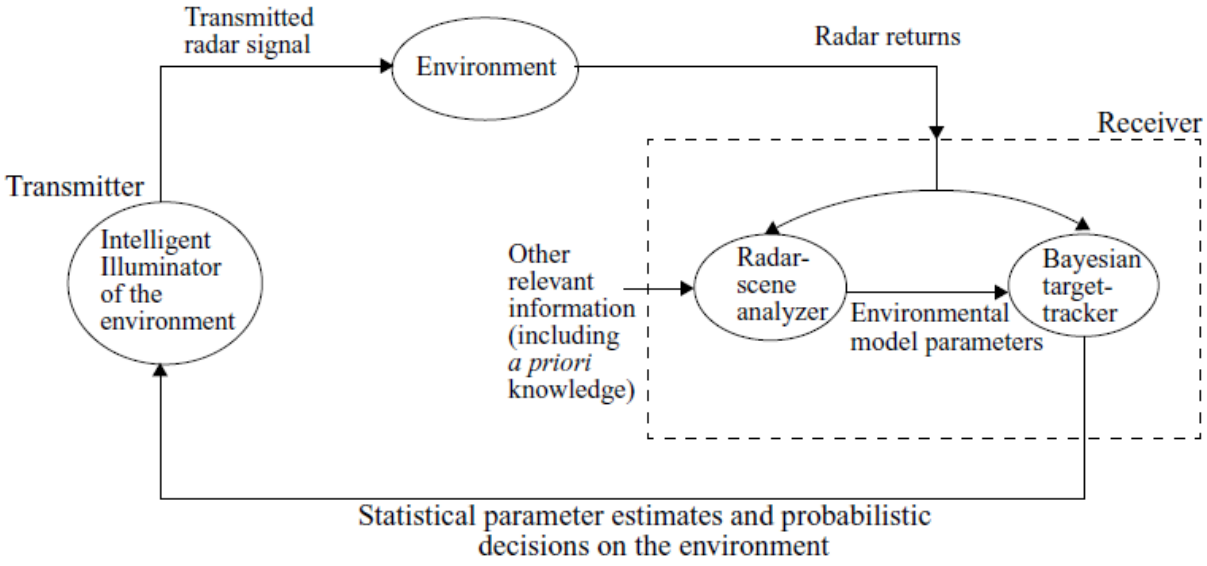


Figure 2-1: Block diagram of cognitive radar closed loop system (Taken from Haykin [40])

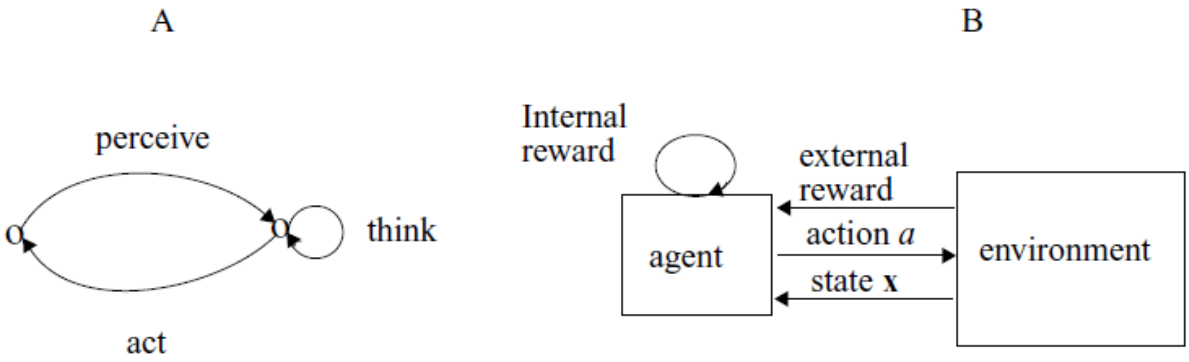


Figure 2-2: Feedback loop (from Haykin)

2.6 Must P=NP for a Universal Problem Solver to be Possible or Useful?

Turing's "imitation game", also known as the Turing Test probes the issue of whether a computational device can become so sophisticated that it can respond indistinguishably from a human to a series of questions [41]. The question precludes that the machine would need to have access to the typical data that a human being possess. However, even if such a machine possessed all this information, could it respond in a manner that reflected human cognitive abilities? Could such a machine learn from the dialogues intended to test the machine and apply feedback ultimately to evolve itself to pass the Turing test?

Some think of the Turing problem as intractable or NP-complete even if a system had all of the data embodied in a human mind [42]. However, this dissertation asserts that P=NP is not a requirement for the design or the useful execution of a universal problem resolution framework

and is not a constraint preventing continuous improvement of the system. There are three main rationales for this assertion:

1) Many optimizations to NP-complete problems derive from observation of data patterns that provide the basis for generalizations implemented via dynamic programming. Examples of this include cutting algorithms and branch-and-bound algorithms that make optimization decisions based on dynamic evaluation of the data associated to an instance of a problem. UPRF naturally promotes optimizations for NP-complete problems by tracking details of all state sequences related to all data inputs. This provides a mechanism to learn from data and ultimately assert rules into the transition state queries that optimize the solutions [43].
2) Approximate solutions to NP-complete problems continue to improve to be effective enough for practical purposes for very large problem instances [44]. For example, a validated-optimal Traveling Sales Problem (TSP) tour for visiting 24,978 cities in Sweden by means of approximation techniques was found in 2004 [45]. Humans demonstrate the use of approximation rather than rote calculation when faced with complex problems. This is evident since a human can accurately guess complex routes to a TSP problem within a small margin of error [46]. Most of the "real-world" problems commonly found in human life are probabilistic and do not lend themselves to direct computation but approximation techniques.
3) By embedding a cost-benefits problem into UPRF and instantiating this as a constraining problem on the system, the framework is able to maintain equilibrium in terms of effort to guarantee maximal effectiveness.

2.7 Problem Solving and Complexity

For many years, computer scientists have wrestled with the Turing test [47] trying to devise methods based on the field of Artificial Intelligence to address the functional capabilities represented by this test. However, all attempts to this point have failed not only in meeting the test [48] (not surprising given the fact that the test requires the computer to be in possession of all of the knowledge associated with a typical human persona), but in demonstrating the type of learning capability needed to achieve that of a human persona. To develop highly complex algorithms, researchers invested great amounts of time to anticipate the wide range of human behavior necessary to meet the Turing test. There have been applications and platforms developed that perform human-like behaviors and able to defeat human opponents in some very complex game scenarios that span beyond typical A/I algorithms including Chess. Deep Thought exemplifies such a pattern, however as is typical of many A/I systems, it is domain-specific, highly dependent on subject matter expertise for training, and computationally optimized for the particular domain thus representing a very narrow scope of human behavior [49].

Despite the advances in A/I, this dissertation asserts that the human learning model is the best software model to achieve the Turing test and that automated machine learning approaches based on the human mind have been inadequate to this point. Nature reveals that the human mind already contains the entire problem solving ability needed to not only potentially solve any solvable problem given adequate data, time and computational resource, but to develop more advanced problem solving automatically through the use of simulation, self-examination and domain cross-over. One only needs to ponder the miracle of human language to realize this. By the age of four, a child has mastered all that one needs in human language to communicate and be understood by

their family without any deliberate attempts to train the child in human language speech and recognition [50].

The focus of AI has traditionally been on the development of specific algorithms or targeting specific domains rather than on the holistic process of learning and problem solving. A holistic process can function across all domains and is not limited to particular algorithms or applications. In the last several years, some noteworthy efforts address the holistic issue of automated machine learning. The basis of these efforts is on study of the brain from both a physical operational perspective as well as a functional aspect.

This dissertation proves by induction and through example that it is possible to model information about problems and the solution paths for problems in a framework to enable eventual achievement of the Turing test. This is accomplished through ongoing pattern recognition that continually analyzes solution paths to generate new patterns which themselves become new problems in an optimal path problem continuum that is domain agnostic. Using a problem definition alone can determine not only the optimal solution paths for a problem, but the optimal solution path technique or algorithm associated with the problem. Such solution path techniques are then employable to optimize the solution for variations of not only the same but also similar problems, as well as for cross training into other domains. The exercise of the solution path techniques become part of the knowledge of the system.

A key issue for such a problem system is the issue of computational complexity. Even if a universal problem resolution framework is constructible, will it be able to operate in polynomial time on P type problems only, or will it be able to achieve polynomial time complexity even within the class of NP-complete? This is a very significant question relative to whether or not the problem

solving apparatus is useful for discovering algorithms for NP-complete problems. This dissertation uses the Tower of Hanoi, an NP-hard problem with exponential complexity having a generalized solution as the base use case. Although the system does not address NP-complete problems specifically, this dissertation asserts that eventually the system will determine the optimal solution path for discovering the optimal algorithm for an NP-complete problem. The system will eventually also improve itself within polynomial time constraints. The problem solver itself represents a variation of the traveling salesperson in that it is seeking out the most optimal path from start state to the end state for its own operation. Even if the optimal path utilizes approximation and the derivation is not deterministic, this does not defeat the utility of the system for generic problem solving and solution optimization.

2.7.1 What is Problem Solving?

For the purposes of this dissertation, problem solving is considered a process for achieving a goal state, given an initial state, including a set of rules providing constraints on how the state may be changed in transitioning from the initial state to the goal state [51]. This definition provides a universal basis for pursuing problem solving in any context that supports state definition and state testing, including mathematics, formal specifications, and even psychology. Within the realm of computational problem solving, a state transition system representable by a Kripke structure [52] provides support for pursuing a solution for a problem. Since states in finite state automata represent values of objects relative to a point in time without specifying the types of objects, this provides unlimited flexibility for defining a state system.

This definition provides the basis for a state machine that can support any arrangement of items in terms of their semantic values related to other objects. Such a definition provides for a

state machine that, for example, could represent a particular equation relative to the variables of the equation or even more significantly the arrangement of a particular problem state relative to other problem states in terms of *solution paths*. For purpose of this work, solution paths indicate a sequence of state changes representing the truth of a conditional state for a problem at each state transition in the transition from the initial state of a problem to its goal state. Another term utilized in the undertaking of solving problem systems in terms of state is *relational state tracking* [53]. Relational state tracking allows a complete history of all state changes within a system. This enables reversibility to a prior state and also supports reproducible reversibility so long as the underlying functions that change state are deterministic and the functional relations projecting the values associated to the states are stored along with the overall data state [54].

2.7.2 Trends in Data Science and Machine Learning

"Big data" has become a buzzword in today's technological world and, although the entire field is buzzing about it, there is no true definition of "big data" at this point. In fact, it seems that everyone has a different definition for it. There are however five key characteristics attributed to this kind of data – whether separately or combined. The first is a volume of data that is large enough to require special considerations in order for storage and analysis. The next is a variety of data that consists of many different types of data possibly also from multiple different sources. Third is a velocity of data production that warrants special consideration – usually in this case the use of old or "stale" data production is not valuable. Fourth is the value of the data produced – i.e. data that has perceived benefit to certain organizations or initiatives. Lastly is the veracity of data – i.e. being able to verify the accuracy of the data [55].

Today's technology news focuses on "big data" and systems that work with such big data. Use cases for forecasting events based on copious amounts of data are increasingly prevalent. Along with this, data science and machine learning are now becoming industry focuses. These trends are all relevant to autonomous problem solving. However, the focus of this dissertation is on a framework that can integrate research from machine learning, data science, and predictive analytics in a holistic fashion. Machine learning systems learn from data for different types of scenarios including supervised and unsupervised learning [56]. The systems serve a useful framework for problem solving by identifying patterns and predicting solutions. However, the emphasis of this dissertation is that no currently implemented systems provide a model for continuous reflection in order to reflect on their own performance and generate higher and higher-level machine learning scenarios that implement improvements into the machine learning systems.

The process for injecting feedback and integrating feedback as higher order problems is not a machine-learning problem but a software design challenge involving recursion, data modeling, and a re-entrant methodology for turning each level of problem solving into an optimization problem [57]. Therefore, this work does not concern itself with machine learning, but rather the framework that can host machine-learning algorithms, inspect their success, and utilize machine-learning algorithms to learn about the process of problem solving. The outcome of this is the enabling pattern matching of higher and higher order problems instantiated into the framework. For purposes of the dissertation, machine-learning algorithms are black boxes evaluated in terms of success and relevance correlated to their outputs and input data from problem spaces, with the algorithms themselves treated as data points in sequences converging to higher levels of learning.

2.7.3 Recursion

Recursion solves problems that are either too large or too complex to solve through traditional methods. Recursive algorithms work by deconstructing problems into manageable fragments, solving these smaller or less complex problems, and then combining the results in order to solve the overall problem. Recursion involves the use of a function that calls itself in a step having a termination condition so that repetitions are processed up to the point where the condition is met with remaining repetitions from the last one to the first [58].

Mathematical induction proves recursion. The definition of primitive recursion is [59]:

I-algebra (X, in) admits simple primitive recursion if for any B ∈ B and morphisms

d; : F;(X x B) -+ B, there exists a S : X -+ B such that the f. d. c. for i ∈ I.

Corecursion then is the dual form of structural recursion. While recursion defines functions that utilize lists, corecursion defines functions that produce new lists. Thus, with corecursion, output rather than input propels analysis and is able to express functions that involve co-inductive types. Corecursion originally came from the theoretic notion of co-algebra with practical implications for higher order problem solving needed in a continuous learning framework [60].

Four primary methods prove corecursion: These are fixpoint induction, approximation lemma, co-induction, and fusion. Fixpoint induction is the lowest-level method, primarily meant to be a foundation tool. Approximation lemma allows the use of induction on natural numbers. Co-induction looks directly at the structure of the programs. Fusion is the highest-level of these four methods and allows for purely equational proofs rather than relying on induction or coinduction [61]. This dissertation utilizes both coinduction and fusion with the Tower of Hanoi to prove that the system supports corecursion in order to achieve continuous improvement.

2.7.4 The Tower of Hanoi as a Typical Problem

The Tower of Hanoi [62] meets the requirements for a recursion problem, although it can be solved through iteration as well. The Tower of Hanoi represents a simple problem that contains a definitive solution pattern for the optimal number of steps. As the number of discs increase, the same solution approach applies in a recursive fashion. While it is trivial to implement an algorithm to solve the Tower of Hanoi, it is not so trivial to discover the algorithm in a generic fashion using simulation alone without pre-knowledge of the algorithm. As a typical recursion problem, the patterns that emerge from the solution model for discovering an algorithm relates similarly to any other recursive problem. Thus, the Tower of Hanoi provides a useful example for an exercise in algorithm discovery. The complexity of Tower of Hanoi also provides an example for incremental domain learning by means of increasing the complexity through adding another peg or by making the number of pegs a variable.

2.8 The Role of Feedback in Problem Solving

It is becoming increasingly more common for computing systems to incorporate feedback in order to provide improvement to software. A simple example that Microsoft Windows™ users are familiar with is the feature that prompts the user if they wish to communicate information about an event causing an error back to Microsoft. By gathering, the information related to the error, Microsoft is then able to try to diagnose a root cause and potentially provide an improvement to resolve the issue back into the software through a service pack. Feedback is a fundamental tenant of evolutionary theory in that it provides a means to incentivize an organism to change its behavior if the feedback is adverse to the organism's survival [63]. Humans regularly employ feedback in their thought processes and it is a fundamental concept for motivational learning theory. Software

evolution based on a feedback loop is a key aspect for continuous improvement within a software process [64].

2.9 Relational Models and Neural Networks

State Sequencing involves the capture of state changes occurring to an object or the attributes of an object. Through the capture of all object and attributes related to other objects and attributes in a system, relational state descriptions are obtained that represent the state changes and their relationships to changes of other attributes within the system. Capturing these relational state sequences allows a complete representation of a functioning system including the ability to replay the model and analyze sequences derived from execution of one model to that of another model.

Relational state sequencing is a key enabler for representing Bayesian networks in both probabilistic and deterministic problems [65]. Storing these sequences into a repository and correlating them back to the source endeavors foster both unsupervised and supervised learning for intelligent analytic systems [66]. Functional programming supports lambda calculus and pattern matching to integrate relational state descriptions in an evolutionary manner to solve problems [67].

Neural networks [68] and relational models have different approaches, but are compatible for information representation. A neural network can be represented relationally and a relational model represented in a network model [69], [70]. This allows the use of a relational model to store the concepts associated with a network learning exercise. Integration of functional programming outputs into relational state sequences that represent the complete behavior of a system allow convergence to recursive relations to generalize behavior of systems [71].

The integration of neural network information into relational sequences within the context of a descriptive relational model enables the relational state tracking approach alluded to earlier in this chapter. Through relational state tracking, the capture of complete information about a system behavior occurs which then enables the complete analysis of the correlations within the system. Through a recursive framework, actions used to analyze the relational sequences become enablers for higher and higher order problem transformations that optimize not only the base problem but the higher order problems of how the framework itself can achieve new capabilities through its own introspection [72]. These new capabilities form the basis for the generation of new algorithms that provide further improvement within a software framework [73].

2.10 Domain Crossover and Associated Limitations

One limitation of many A/I systems is that their orientation is focused to solving a problem in a particular domain such that the formulas, functions, data structures, and other aspects of the software application are optimized for the domain and may have limited utility outside the domain. For example, while neural network architectures for single-layer nets have shown relevance to certain domains, neural networks for multi-layer domains suffer from interaction complexity [74]. An algorithm applied effectively in one problem domain may be completely ineffective for another domain. This is because problems vary significantly in terms of their initial state configurations, types of objects involved, transition state constraints, and final goal states. Each of these has an impact on what technique is optimal for the particular domain.

The lack of automated algorithm selection depending on the problem type is a common problem encountered by novice users using sophisticated data mining software. Many data mining packages prompt the user as to the type of mining model to use, whether it be associative,

clustering, structural, pattern-based, time-series, etc. Often the users of this system lack the subject-matter expertise or experience in the domain to make a well-informed decision. Some systems now try to predict the model that will be most effective by sampling the data and evaluating the effectiveness of a model, thus incorporating feedback as part of the problem solving approach.

One approach commonly used for problem solving in traditional A/I is the use of genetic algorithms [75]. NP-complete problems domains commonly use such algorithms as well as other domains with some success. Genetic algorithms can have utility for cross-domain learning as a means to optimize a solution approach rapidly [76]. However because genetic algorithms focus on energy functions for a specific domain and follow an evolutionary model that optimizes toward a particular goal state in a random fashion, their usefulness for finding the best solution that can map from one domain to another is limited, particularly if genetic drift accommodation is lacking [77]. Through an approach that utilizes machine-learning algorithms, including genetic algorithms as black boxes inside of the holistic framework, the genetic algorithms themselves along with their limitations and correlations to problem types become an area of continuous learning within the framework.

Domain crossover for the context of this dissertation indicates a deliberate effort on the behalf of a system to apply a solution technique to another domain in order to determine the usefulness of the technique to the newly targeted domain. A simple way to determine this is to actually test the technique in the targeted domain and evaluate effectiveness relative to other techniques. However, this technique imposes potential inefficiencies – one of which being the difficulty in determining an effective sample size that adequately tests the technique since the data values may

be skewed for certain attribute values for entities subjected to the technique. In addition, even if a technique applied to a particular problem in a domain is effective, how does one know that it will be effective to other problems in the domain? For example, what are the criteria that cause a technique to be effective that may vary even within the same domain for different problems?

Making decisions without all the information on a system is a challenge for decision systems and expert systems. Often subject matter experts (SMEs) train these systems to orient data mining techniques based on certain conditions in the data or types of problems; however, this is a tedious and error-prone process. UPRF's ability to automate the trivialities of inspecting data results against a goal reduces SME interaction.

When contemplating the various obstacles of domain-crossover, it may seem that attempts to utilize proven techniques from another domain or from dissimilar problems have such a low likelihood of success as to be virtually no improvement over random selection -- perhaps even worse considering the addition effort of applying techniques likely to fail. However, this assertion is proven false by examining systems that are effective and do utilize domain-crossover. A good example of such a system lies in the area of human learning and problem solving. In human problem solving endeavors, people often successfully apply a solution technique from a seemingly unrelated domain onto another domain.

How do humans accomplish domain-cross-over and can the same techniques apply to software-based problem solving? For example, how is it that a child who gets bit by a dog as a by-product of harassing it make the decision to be careful in treating other types of animals with no resemblance to a dog? In this case the answer lies within the question of whether the child is able to recognize that there are commonalities between animals such that a dog and a cat are both related

closely enough that the cost/benefit of restraining their impulse has a positive return [78]. For example, a young child who is bit by a dog because of treating it roughly, with limited knowledge of machine commonality from dogs to self-propelled vacuum cleaners (i.e. a Roomba iRobot™) may hesitate before treating a vacuum cleaner roughly. These examples indicate correlation is the enabler for human achievement of domain crossover. In the same way, problem-solving systems should also be capable of domain crossover by linking attribute commonality (either directly or indirectly) among problem domains.

2.11 Association, Pattern Matching, and Prediction

Based on the concept of associative correlation and commonality [79], an equation can be constructed to avoid an adverse response wherein the initial state involves an object that relates to another object belonging to a problem that has already been solved. Combining the concept of feedback with association enables prediction generation. In "The Two-Second Advantage" the authors postulates that humans make decisions based on attempting predictions and monitoring the success rate in response to associative patterns [80] and thus optimize their effectiveness to adjust to new circumstances. Without this ability, humans would not be able to function because the amount of sensory input received by the brain is too much to analyze completely in time to respond effectively. Consider the millions of neurons that are fired every 40 milliseconds or so from images transmitted by the retina in comparison to the number of computations that a person can perform in one second [81].

2.12 Fuzzy Logic, Probability, and Statistics

A study of predictive learning links to the concept of "fuzziness". Computation commonly focuses on binary absolutes wherein a logical expression yields either true or false with some systems including the concept of undefined/unknown. However, decision-making as done by organisms, particularly human is not strictly binary, it is based on probability. Probability is a key aspect of data mining in order to determine correlations. Probabilistic data mining systems estimate likely correlations. The more robust data mining systems calculate confidence intervals that accompany the correlational findings. Utilizing "fuzzy logic", programs are often able to achieve higher efficiency while retaining high reliability for the targeted problem [82]. Recommendations result from probability models, which while not guaranteed to be correct, are likely enough to justify acting upon probability calculation. A good example is a high-frequency trading application that does not have time to examine all of the data, but has enough time to identify significant deviations to anticipate a market risk and perform a mitigation action. This concept is useful for mathematical models including Fourier Transformations to reduce problems of unsolvable scope via estimation or division to probability problems that reflect the accuracy of the fully constituted problem accurately enough to be useful while not being too slow to be practical [83].

2.13 Holistic A/I or "Authentic Intelligence"

At this point, it is apparent that the optimal combination of logical techniques performed in an optimal sequence is greater than the sum of its parts. That is, each of these techniques taken to an extreme without integration to other techniques has less value than the **intelligent** combination of the techniques. Pure prediction is nothing more than guessing with limited utility. Calculating statistical likelihoods is a waste of resources for a simple problem. Constantly changing a solution

technique due to occasional unexpected feedback is worse than relying on one that is usually correct. Associating items that have similarity into the same problem class is not useful if the similarity does not relate in some way to determining the outcome of a solution technique. Even reliance on pattern recognition and relational state tracking may lead to false solution paths if the relational state sequences form the same pattern for a certain grouping of initial configurations. For example if Tower of Hanoi is instantiated only having an odd number of disks, pattern recognition will yield a technique that dictates always moving to the third peg on the first move. This solution pattern will prove faulty once the initial state is set for the game to have an even number of disks. Until a suitable variety of test cases establishes a causality relationship between the number of discs and the required first peg move, the apparently universal truth condition arising from a preponderance of test cases will be in a state of failure.

Based on the above, the optimal problem-solving model does not confine itself to a singular approach, but is combinatorial in the selection and sequencing of application of various techniques to a problem. This brings the focus to intelligence. In order for all these techniques to work together, the techniques must integrate in the most optimal fashion. In this sense, traditional A/I work is mainly too narrow in scope. The emphasis tends to be on particular techniques such as neural networks, Bayesian-based, symbolic logic, pattern matching, etc. (all of which work well when paired to the suitable problem types) rather than on the problem of identifying how to find the optimal combination of techniques for a problem solution.

2.14 Self-Organizing and Diminishing Returns

The concept of searching for solutions to problems in a symbiotic fashion that benefits the overall system manifests in the principle of self-organization [84]. Self-organization is a

requirement of UPRF in order to improve itself without external modification. To be effective, self-organizing systems must possess the following attributes [85]:

- Autonomy: The system needs no external controls.
- Emergence: Local interactions induce creation of globally coherent patterns.
- Adaptability: Changes in the environment have only a small influence on the behavior of the system.
- Decentralization: The control of the system not done by a single entity or by just a small group of entities but by all entities of the system.

Merely finding the optimal way to combine techniques for a particular problem and then using them is of little utility given the myriad types of problems and techniques available. Such an approach if done randomly is undoubtedly worse than using any one technique exclusively. Therefore, a more important problem is discovering how to find the optimal solution combination technique. However, even if one solves the second order problem of discovering an optimal solution combination that finds the optimal combination for the lower order problem, can the second higher order problem for finding the optimal combination also be optimized? One answer is to transform the second order problem to a higher-level problem targeting the goal state as being the optimal way to find the optimal way to find the optimal way to solve the original problem.

2.14.1 The Infinite Recursion Problem

The dissertation asserts that any problem whereby complete state tracking of all the steps utilized to solve the problem including capturing all of the function calls, assertion tests, and outcomes can be transformed to a higher level problem that is representable in a generic state machine. However, even with implementation of such a means for recursive problem optimization,

the perfect solution path will never be determinable since each level of solution path optimization leads to a higher order problem. The outcome is that each higher order problem targeting the optimization of the lower order problem provides additional benefit albeit with diminishing returns at some point. Another problem is that if the recursion loop fails to unwind, none of the problems in the optimization hierarchy ever receives any optimization. Constructing such a system for solving problems would in fact never solve any problem because it would never exit the recursion loop; this is known as the *infinite recursion problem* [86].

2.14.1.1 The Recursion Exit Dilemma

For recursion problems that involve potentially infinite recursion, a constant or type of expression serves as an exit condition to cause the unwinding of the recursion [87]. This technique has application to the recursive problem solving machine construction outlined earlier. This seems the best option with UPRF since infinite recursion is of no use and the framework can provide another optimization problem that concerns the exit condition for the recursion itself.

Depending on the problem complexity and effectiveness measured at each iteration of solver recursion, the optimal exit point is highly variable. Therefore, use of a constant to control this is inadequate. A better approach is to unwind the recursion in parallel spawning sub-problems until exploration of the solution space for the lower order problem completes or reaches some threshold. In this model, co-recursion occurs with recursion such that the new high order problems derive from lower-level instance solving and optimize the lower level problems. In this scenario, the optimization techniques are passed down to the lower level problem immediately or deferred by a recursion gap representing a threshold of hierarchy levels that triggers the application of the techniques back to the lower level problem. One advantage of the co-recursive/recursive approach

is that the system can infinitely attempt to optimize higher and higher order problems while providing utility and continually improving.

From the above, continuous improvement capability is possible provided the constraints of generic problem representation, complete relational state representation, and a base set of functions have sufficient robustness to accommodate analysis and problem solving. Sufficient robustness in solving and analytical functions is achievable through at least three different means:

1) The functions contained in the baseline of the system provide a high enough coverage for pattern matching, probability functions, data mining functions, condition assertion functions, and condition testing functions such that the exercise of the functions in concert and tracking of their invocation patterns is better than random choice;

2) There exists an extensibility function such that the system is able to incorporate new functions autonomously in response to dead-ends for solutions using some type of dynamic function execution model combined with dynamic search or an ability for a SME to add functions dynamically (however, this is a violation of the principle that SME intervention should not be needed);

3) The system is able to write its own functions, track the primitive operations used for writing the functions in terms of relational state sequences relative to the optimization and dynamically execute them by combining primitive functions in different combinations in order to accelerate the effectiveness rate of function application relative to optimal solution paths.

2.14.2 The Equilibrium Paradigm

Equilibrium within a system manifests when the system reaches a steady state or ranged state such that a balance between different functions exists [88]. This result is typical of systems that

act upon themselves; the reactions in the system arise from actions and have a counter effect on the initiating actions over a sequence of states. Financial markets to a certain degree exhibit equilibrium since money inflows, money supply expansion, and money redistribution act upon the overall system [89] (Pareto Principle [90]). Equilibrium generally indicates that a system has reached a level of optimization whereby alterations do not provide benefit if the system is achieving the desired goal state.

Multi-agent reinforcement learning can facilitate equilibrium in machine learning systems [91]. These systems utilize cooperating agents to converge to a desired equilibrium that spawns actions that contribute in actions that optimize reaching of the goal. Due to the recursive nature of UPRF and the possibility for endless iterations, equilibrium is a key aspect to accommodate in the framework as a continuous operational problem. Utilizing the framework to host this equilibrium problem allows UPRF to leverage the same infrastructure as that used for other problems and reap the same benefits of cross-domain learning and continuous improvement.

2.14.2.1 Targeting equilibrium as the goal state

Based on the above, the master problem for a universal problem solver is to achieve and maintain equilibrium where equilibrium represents the optimal ratio of computational processing utilized for optimizing the system relative to computational processing utilized for solving problems. Herein lies a central tenant of this work; the optimal solution to a problem is able to be defined in terms of reaching equilibrium in a co-recursive/recursive problem solving system across all depths of the hierarchical problem space including the optimal method for arriving at optimal equilibrium. By targeting equilibrium, the framework avoids arbitrary recursion exits and is able to manage its recursion.

The concept of self-reflection was alluded to earlier in the context of a feedback loop and as a differentiator of UPRF compared to existing problem solving frameworks. This self-reflective process ensures that measurement of all actions is relative to prior results for correlated actions. In some cases, the correlations may be distant but still relevant. For example, one may consult the Internet to obtain directions and a time estimate to travel to a specific location but the time estimate is only accurate given the current traffic. The estimate may not be accurate for a later time when some event may occur that adds to the traffic. A person may mistakenly not factor in the occurrence of an event and assume the travel time to be exactly as prescribed on the web site. It is only later when the person arrives late due to some large event occurring along the traffic route such as a sporting game that the person learns to not only search for the directions and estimated time, but also for nearby events related to the destination. This type of learning affects our processes even across domains to the point that the experience may remind of the need to be cautionary in a seemingly unrelated endeavor such as finishing a group collaborative document. The experience of lateness due to an unexpected event may heighten the fear of being late on a project involving multiple individuals where the factoring of the commitment level of all individuals is not included. This process of self-reflection ultimately leads to more complex and interrelated schema from a human perspective [92] as well as when implemented in software.

The targeting of equilibrium is a natural goal state postulated by evolutionary theory and exemplified in human learning. The human mind is capable of endlessly exhausting computation cycles in self-reflection to a point that the individual takes no action resulting in a paralyzed state. Yet, without this model of reflection, there is the risk of an impulse decision that does not leverage experiential knowledge leading to a disastrous error in judgment. Exclusivity to either extreme is

disastrous to self-preservation and prevents survival. Based on this, it is evident that the human organism through decision-making driven by learning and problem solving experience is in a constant search for equilibrium. Thus, a system that targets equilibrium in self-reflection and optimization (self-organizing) in terms of higher order problems to generate optimality patterns for lower order problems and continually probe cross-domain problem solving improvement as a relation onto its exercise of computation for direct problem solving should be achievable. It is also arguable that such a system emulates human creativity since creativity appears to arise out of pattern recognition between distantly related problems triggered by a commonality relation that if true would drastically improve the solution for the problem. The concept of creativity in this context represents the outcome from pursuing an optimization goal driven by the probability that a decision with potentially low likelihood to succeed is worth attempting based on the metrics of prior forays.

2.14.3 Managing Time and Storage Complexity

The time and storage complexity of a problem solving system is exponential for NP-complete problems until optimizations are found, since the problem space must be explored as a search problem that exponentially increases in size for trying out all possible state sequences [93]. However, the equilibrium goal places a constraint on the system for optimality that requires the system to retain polynomial time complexity. This allows the system to constraint itself to finding optimizations that may not be exact but are justified based on a return-on-investment principle where a "good-enough" solution is better than no solution at all.

2.15 Towards a Generic Schema for Problems

For a problem solving system to function generically across different problem types, the system must be able to interact generically with all types of problems. This implies the need for a general-purpose schema for representing information about any problem. Historically, problem domains rely on semantics and schemas specifically targeting the domain rather than on generic constructs. Various models have been put forth to try to make schemas for problems generic [94], but without a unified model from that can represent knowledge from all problems, integration of knowledge across problem space requires custom agents that can interpret the information specific to a problem domain. Case-based reasoning (CBR) systems provide a model for such a generic database for general purpose problem solving. CBR functionality is dependent on a path and pattern database that stores the all problem and solution states [95].

Proof that such a schema also enables autonomous improvement arises through a simple Tower of Hanoi scenario where multiple variations of the game play out using different instances. The next chapter illustrates this by instantiating two through five discs to generate relational state sequences intersecting different conditions of specific attribute values to conditions defining other attribute values in the system. The relational state sequences incorporate derivative states to provide recognition sequences that reflect problems that project into functions based on change frequencies. Many NP-hard problems exhibit symmetrical patterns when interrogating the problem space related to the solution space including Tower of Hanoi.

In the Tower of Hanoi scenario, analysis of discs two through five reveals symmetrical patterns allowing simple pattern recognition using bit functions to generate functions that generalize conditional expressions relative to the number of discs for setting the peg number. The

conditional expressions that prove as valid transition state optimizations to add to the rule states are able to generate the prediction state sequences by the time that the count of discs reaches six, providing linear time complexity for the solution path identification after the first four test cases. Generalizations spawned from optimizing the higher order problem for finding the optimal solution serve as optimization conditions to add to the lower level problems. The framework revisits the lower level problems and applies the generalizations. As more variations are tested, pruning the set of candidate generalizations becomes possible. Using the tested conditions as optimization expressions starting with a count of discs equal to six, the conditional expressions that prove as valid transition state optimizations provide linear time complexity for arriving to the solution path for all future attempts regardless of the number of discs.

2.15.1 Rationale for using a Relational Schema

Relational algebra provides [96] finitary relations that allow objects to be organized relative to other objects in terms of existence dependencies as well as enabling object values to be associated to the parent object containers. This allows a relational database schema to represent semantically how collections of objects relate in terms of dependencies as well as values. Such a schema provides utility for UPRF since full state representation is dependent on object values relative to each other at each sequence in a problem solving exercise. Since the values are able to provide the linkage between the objects and this information is part of the information schema of the model, this effectively supports neural-network constructs of associating an object to another object. Metrics regarding the relative strength of a connection can be determined in terms of the number of intervening nodes as well as in terms of numbers of physical connections based on cardinality between the nodes.

Value change state relative to other values in the model over a sequence of states yield binary strings indicating truth or falsity between every object in the system for every state. Through application of transform operators successfully predicting the outcome of one instance from other instances, a progression of transformation operator sequences results. The UPRF schema thus demonstrates an effective modelling of a neural network within a relational construct with all objects in the endeavor linked together.

Since the relational model provides for information about itself within the same model containing the information, it supports self-representation and a recursive data structure, which are both foundational to provide generic semantics to agents interacting with the structure. This is a key component for supporting the recursive processing identified earlier associated with a problem solving system that can interact with itself.

2.16 Rationale for the Possibility of UPRF

The human mind is vastly different from current computer systems in terms of structure and operation; however, computer systems and the mind share many similar attributes. These attributes allow computer systems as well as humans to fulfil the requirements of an "Intelligent System" [97]. Intelligent System implies the ability not only to solve problems but also to learn and improve in a continuous fashion with minimal outside intervention. This section examines evidence supporting the thesis that capability for autonomous continual learning present in humans is possible for a computational device through appropriately designed software.

Many have attempted at distilling human problem solving into software frameworks. Among these attempts include the use of cognitive primitives [98], evolutionary multi-agent systems [99], and algorithm pools [100]. UPRF is novel in this regard since rather than choosing a particular

machine learning strategy, it resides at a layer above the A/I realm to act as a holistic consumer and benefactor from the underlying algorithms.

2.16.1 Attributes of Intelligent Systems

This section explores the attributes that support the processes of general problem solving and ongoing improvement or optimization of the solution-discovery process. Four attributes surface as possible cornerstones for understanding a problem, discovering solutions, and profiting from the exercise to enable improvement of the overall problem solving framework so that additional problems incrementally benefit from the problem solving experiences [101]. These attributes link to the processes outlined in Figure 3-1. They are:

1) Problem Representation [102]: A system cannot endeavor to solve a problem unless the problem can be schematically represented such that a solving agent can understand the starting state, desired state, and allow intermediate states that are traversable in order to find solution discovery paths.
2) Solution Space Probing: A solution to a problem is undiscoverable unless there is a process for exploring possible solutions to determine the arrangement of steps needed to achieve the solution.
3) Performance Metric Collection: For a system to be able to improve it must have a mechanism to collect metrics about how well it is performing so that these metrics can be analyzed to determine the relative effectiveness of actions carried out by the system for problem solving.
4) Performance Analysis: There must exist a process that correlate the effectiveness of a system to meet its goals with the steps taken to obtain the goals.

2.16.1.1 Problem Representation

A generic solving system is only possible if the organization of problem information is in a structure that supports generic evaluation. Problems that can occur in wide varieties of domains involve a wide variety of different types of objects. It is impossible to anticipate beforehand all of the different types of problems that can occur. Therefore, self-improvement and continuous learning require that the problem-recording structure is robust and extensible enough to store information about a problem even without knowing the manner of it solution.

In humans, children exemplify this paradigm in language learning. Children receive language information well before the time they are able to use language to express themselves. Despite the inability of an infant to understand spoken language or use it in early infancy, the capability to decipher speech and learn to use it develops autonomously through experience without deliberate instruction. Research suggests that the child is able to schematize information about language even before it is able to decipher the meaning and semantics necessary to employ language [REF].

How humans are able to schematize problem information such that a solution discovery process can understand the meaning of the problem and work toward a solution is a mystery. One theory is that neurons and synapses form a neural network in the brain through connections that represent information about a problem [103]. Neural networks provide a method for representing information through connections. The arrangement of a set of objects perceived can thus map into a network that contains the information about how the objects inter-relate to each other. An unanswered question regarding human thought concerns how the mind is able to map particular types of connections to thought processes associated with thinking about and solving the problem.

Software applications provide a caricature of problem understanding and solving for specific domains using a variety of different methods for representing information and then processing the information to meet the goals of the system [104]. For example, a flight reservation system stores information about flights, routes, and scheduling constraints such as airplane seating capacity, and factors affecting efficiency and performance of the aircraft including speed, range, fuel usage, etc. Advanced flight dispatching systems not only allow the flights to be scheduled, but optimize the scheduling to meet some target goals such as clustering flights toward a particular hub, minimize under-utilization of seats, and maximizing the likelihood for on-time arrivals. An intelligent flight reservation system is only intelligent because the software has been oriented to pursue goals defined as part of the data.

Operating systems also provide an example of software that attempts to optimize its own performance through heuristics [105] and rules that define the goal behavior [106]. Operating systems seek to minimize resource deadlocks, assure maximum throughput, and maintain optimal equilibrium in the use of memory, storage, and computational resources. Relational databases also include mechanisms such as I/O cost evaluation in order to attempt to optimize queries for maximum throughput [107].

These examples provide evidence that it is possible to build intelligent systems within particular domains. Combining data structures that represent the state constraints of a problem with processes that interact with the states to perform simulation and correlational analysis provides a problem resolution framework. This evokes the question of whether or not it is possible to abstract the domain-specific details of computational problems in such a way as to promote a general-purpose problem-solving framework.

The traditional approach for solving problems by focusing on aspects of the domain does not promote generic problem solving and modeling, thus compelling the need for another approach to achieve optimality without domain-targeted programming. Is it possible to model domain-specific information concerning goals purely declaratively as the data relates to functions over a sequence of time rather than in imperatively terms of procedural code? A key to finding these answers lies in distinguishing declarative and imperative program definition. Imperative programming approaches utilize a sequence of steps to solve a particular problem. Declarative programming implies that computational logic follows as a derivative from rules that define the inter-relationships of structural and relational data objects. Relational databases are a form of declarative programming in that much information about the computational requirements for a set of entities can be defined through the relationships specified in a relational database [108]. For example, a database relation can enforce the requirement for a customer to exist before creation of an order. The relational approach of UPRF promotes a declarative model wherein the data does not only include the details of a particular system, but information about how the data within the system works together to define it. Data that provides information about a system's information constitutes metadata. Comprehensive metadata is a key foundation for enabling generalizing information about problems such that solutions can be data-derived rather than procedurally prescribed.

A second key enabler for generalized processing that extends beyond domain specificity is the use of polymorphism. Polymorphism allows management of different types of objects through the same interfaces [109]. In the case of the flight scheduling system and an operating system, a polymorphic system can implement common functions for optimizing performance and efficiency

despite the differences in the data structures. Polymorphism allows for abstraction of functionality from structural data implementation details.

Generalization is a third aspect for enabling information representation that facilitates the abstraction of data from processes [110]. Generalization allows for aggregation of multiple aspects of information to in terms of a higher-level concept. Object-oriented semantics such as generalization and abstraction, as well as relational representation, are information storage and processing mechanisms that undergird problem representation and solving in human thought processes, also existing as well-defined software design patterns.

Using object-oriented and relational constructs along with state representation for declarative programming ensures that information about problems lends itself to a generic representation structure through multiple levels of abstraction. UPRF provides semantics for generalizing problems in terms of their objects in a relational manner relative to the pertinent states associated with object configurations needed to navigate from a problem definition through intermediary steps to meet a goal state. The dissertation outlines implementation of these constructs through a relational database schema exposed through XML schemas that demonstrate the utility to support universality.

2.16.1.2 Solution Space Probing

Assuming a common semantic model defines a problem based on relational representation, will the resulting problem definition support generic solution approaches? Probing of solution paths associated to multiple instance of a problem and transforming correlations learned asserts this attribute of human problem solving is available to a software framework. As the framework

catalogues higher layers of problem solving abstraction, progress toward solving increasingly complex problems occurs.

Functions that control the possible solution paths provide the constraints for solving any problem. Possible solution paths derive from the outputs of functions underlying the rules for the solution space. Together the functions that define the potential solution possibilities along with the ranges materialize the potential states of a problem related to its solution. Turning back to the generic representation concept, if a problem has been defined in terms of initial state, goal state, and allowed intermediate states in the context of encapsulating functions, then it follows that a generic solution probing process can explore the potential states and branch the problem states until ultimately achieving the goal state. Chapter 4 provides a walkthrough of how this is accomplished.

2.16.1.3 Performance Metric Collection

A system that can generically represent problems and automatically probe for solution paths may be useful but it cannot allow for intelligence without a mechanism for optimization. Optimization requires metrics [111]. Improvement requires a baseline and a target. Optimization is in fact a problem in and of itself with the initial state being a brute-force solution discovery to meet a goal for a computational problem and the goal state being the optimal sequence to find the steps for solving the problem. The intermediate steps are the rules that the system can follow to attempt to achieve the optimal path. For UPRF to be able to accomplish this, capture of all actions performed by the framework must occur to support correlation of the actions to the effectiveness of reaching any particular problem's goal state.

The aspect of collecting and analyzing information related to a process is an area of knowledge management in data mining increasingly implemented in more and more software applications and business processes. Examples of such systems today include applications for counter-terrorism, insurance actuarial analysis, online shopping-basket customer sales prediction, and financial risk management. Therefore standard data mining, including collection, analysis, and interpretation of data related to operation of UPRF is a fundamental component to enable the continuously improvement mechanism. A weak point in most data mining implementations is the requirement for a subject matter expert (SME) to examine the results and provide interpretation. However, by transforming the analysis process into just another problem with a baseline state and a goal state as is done for all other problems, the optimization process itself can be enabled for automatic improvement.

2.16.1.4 Continuous Improvement

Continuous improvement [112] is a differentiating aspect of UPRF achieved by transforming the problem of solving a problem to be a problem in and of itself. This is where the recursive aspect of the system comes into full force; it allows the system to view the problem of its own performance as just another problem within the same framework as any other problem. This design ensures the system is unlimited and not bound by any pre-determined abstraction definition. The functioning of continuous improvement is the most advanced aspect of UPRF. The continuous learning aspect functions in an autopoietic method capable of self-enhancement without enforcement from a supervisory external force [113]. This is in contrast to allopoietic systems that are inherently fragile and require continuous intervention to maintain operation and implement improvements.

2.17 Chapter Summary

This section reviewed the literature associated with computational problem solving across a variety of disciplines including Neuroscience, Artificial Intelligence, Machine Learning, Software Architecture, and Information Theory. This detailed study drives not only the requirements for a universal problem-solving framework (UPRF) but also points the way to key principles and concepts needed to support such a framework. From this detailed exploration of these areas, several key requirements emerge that UPRF must address. These requirements include:

1) A generic representation of a problem including the queries that associate functions and data attributes to generate objects that map to its initial, goal, and allowed transition states;
2) A system that can probe the solution space based on the problem states in a general fashion without knowledge of the problem domain;

   3)A mechanism to learn from the solving of related instances in order to create higher and higher level abstraction problems that yield higher level instances, which generate higher order assertions that can ultimately generate solution paths without simulation.

3. SYSTEM FRAMEWORK

3.1 Framework Foundations

Computational problems may be in various types and forms, but there are qualitative aspects and attributes common to all problems. This section identifies these common attributes and establishes constructs and semantics for the generic definition and solution of problems. These constructs then guide the design for UPRF. This section provides an informal proof augmented by examples to establish credibility for the feasibility of a universal problem resolution system that continually learns.

Before construction and design of any software framework, one must define the entities that constitute the actors, objects, interactions, and boundaries. In addition, the designer must identify the use cases that include scenarios of operation to define the interactions and rules that the framework must support. Finally, the designer must determine the critical success factors that include the goals and non-goals to ensure the framework meets the design requirements. Frameworks are not applications in and of themselves but rather provide the infrastructure for hosting applications to achieve a purpose [114].

UPRF's goal is to support problem probing, solution discovery, and solution optimization in a general fashion. The framework is able to represent any computational problem in terms of its objects with queries mapped to functions to identify starting state, allowed transition states, and a goal state. The states may be complex and involve combinations of sub-states. There may also be

states to represent constraints about the solution process including those that limit resources or associate to some condition in the solution process governing the extent of effort for pursuing the goal state. The framework allows for simulation problems that do not include an explicit goal state in which case the goal state is simply to exhaust all possibilities in the scenario.

Various components work together with a schema to achieve UPRF goals that support both generic schematization and resolution of problems. This section provides the basic framework design that enables operation. The focus of the work is not on the framework construction in and of itself but only as it pertains to enabling operation of the problem solving process. Therefore, this section prescribes the high-level components only. Chapter five on practicality delves into more detail regarding implementation scenarios and maps the framework to technological trends.

3.2 Base Use Case

Tower of Hanoi serves as the main use-case for this dissertation. Hanoi provides a good use case because it is possible to pursue multiple solution paths for multiple instances of Hanoi generated from the number of discs used. Tower of Hanoi only has a single optimal solution for each instance that simplifies the presentation of the concepts. It is important to recognize that the choice of Tower of Hanoi does not limit the design of UPRF; the framework supports any type of problem regardless of the specific features of Hanoi. Chapter 5 explores the ability for UPRF to scale beyond Tower of Hanoi to other problems.

Tower of Hanoi itself is NP-hard, but the process for discovering a general solution for all instances to this – or any other problem (the theorem proof problem [115]) is actually NP-complete. In other words, even though it takes exponential time to solve an instance of Hanoi related to the inputs, finding the solution for a general case is still NP-complete. After the

transformation process yields a generic solution, application of the general algorithm discovered proceeds within polynomial time. The assertion from this is that the NP-complete process for discovering optimal algorithms stands to improve through monitoring and learning from all steps involved in the optimization. Since solution discovery is in the family of NP-complete, then all NP-complete scenarios stand to benefit and should be helpful for any solving endeavor whether the algorithm for the problem itself is NP-complete, NP-hard, or P.

3.3 Core Architecture

UPRF must support a problem transformation paradigm where each solution sequence becomes the target of a higher order problem transform in order to generate increasingly generalized transformation sequences in search of general solutions to lower instances. Figure 3-1 illustrates this concept - each solution determination spawns a higher order optimization problem to generalize the steps required for multiple instances of a lower level problem. In the case of the Tower of Hanoi, each circle represents the higher order problems targeting the instances of the lower order problem. At the lowest level, multiple solutions target different disc-counts. The solution process generates sequences that an inspection process uses to determine how to transform from a simpler instance to a more complex instance. Chapter 4 iterates through this process in detail.

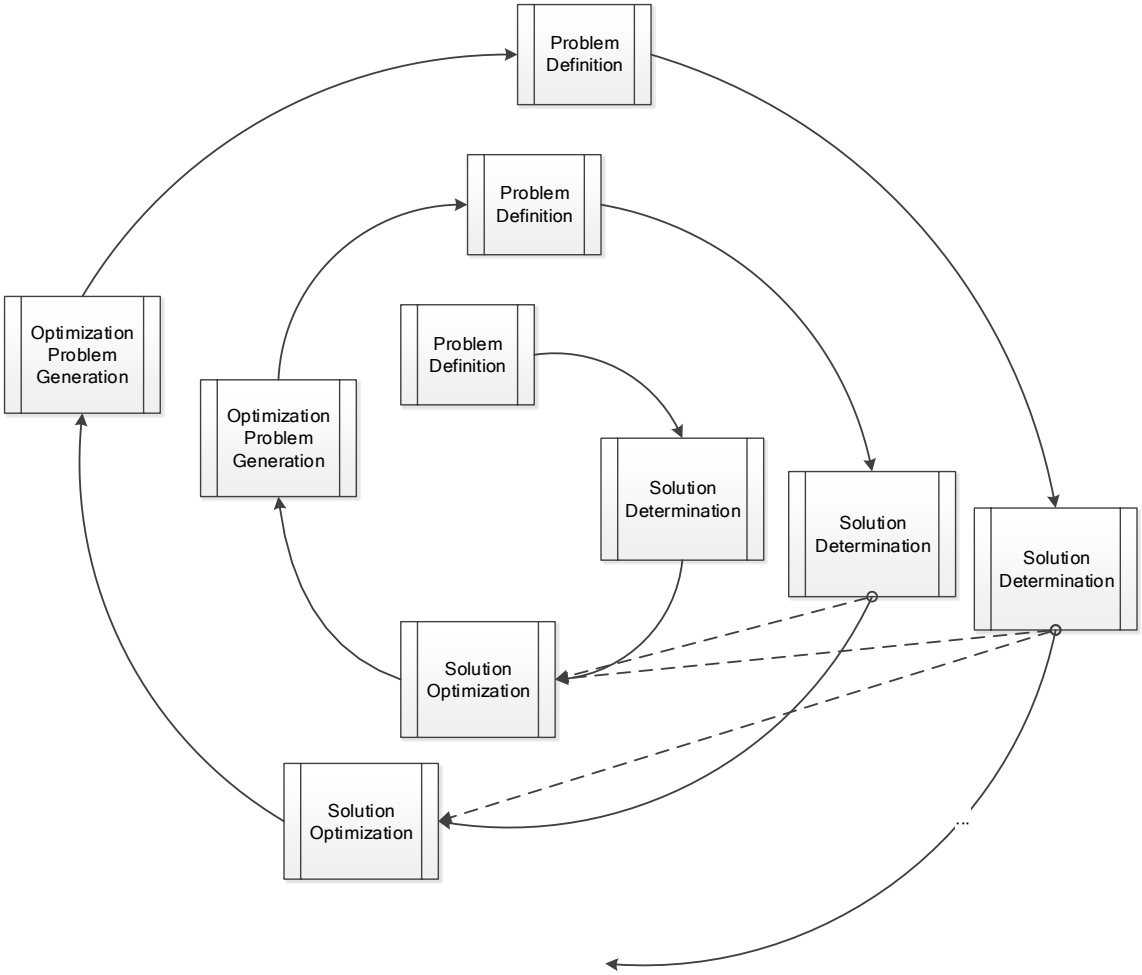


Figure 3-1: Continuous Improvement Cycle

In Figure 3-1, each solution determination step transforms to a higher-level optimization problem with the target of determining the optimal steps needed to generate the steps for the lower level problem. The lowest level of problem solving is pure simulation using search based on the heuristics of the problem that govern the valid states. When there are multiple instances for a particular problem, as in Hanoi, UPRF spawns a transformation problem to attempt calculation of the sequence of steps for one instance based on one or more other instances.

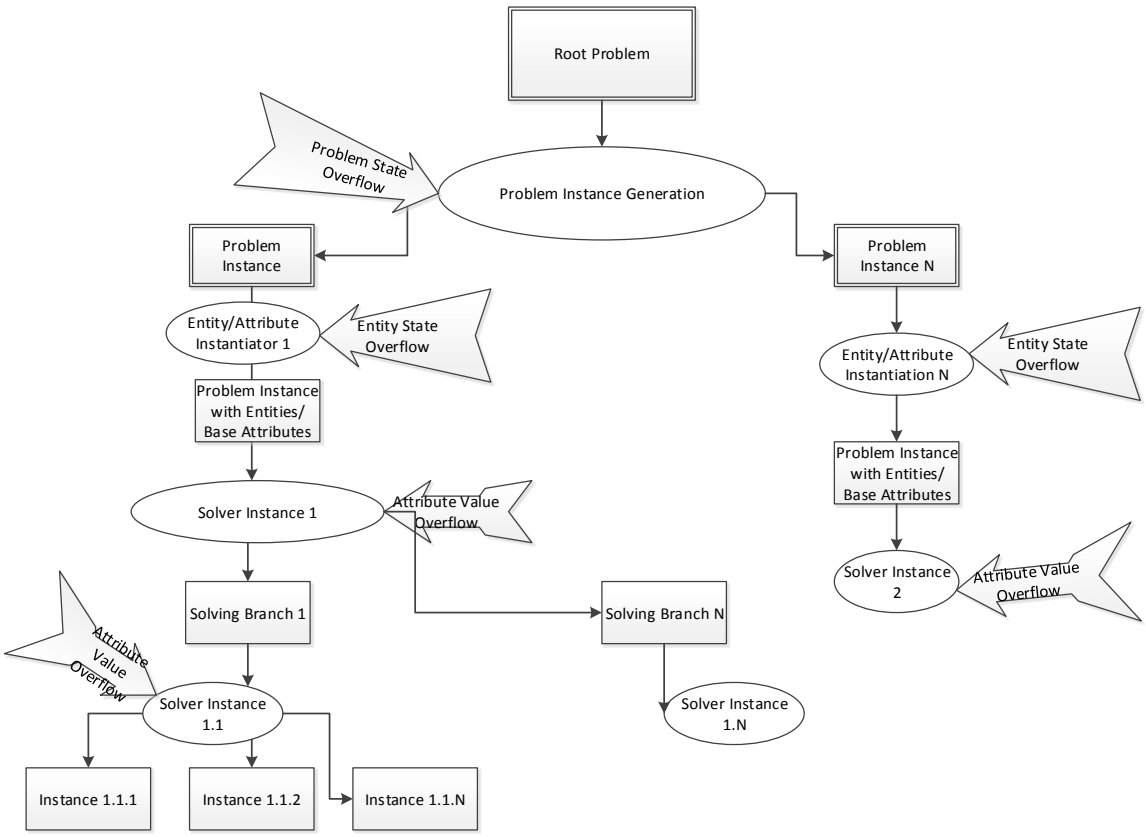


Figure 3-2: Simulation Process

Figure 3-2 illustrates the approach for exploring possible solutions to a problem. UPRF queries for the allowed setup states when first initializing a problem. If there is more than one instance, the additional instances are branched from the main instance for parallel solving by the engine. This is the case with the Tower of Hanoi as shown in Figure 3-3; different branches develop as the simulator tries to work through the solutions for a particular number of discs. The variable number of discs overflow the problem instance space creating *problem state overflow* in Figure 3-2 generating the required instances.

Problem state overflow also occurs when multiple outputs arise at some level in the simulation. This when multiple attribute values are available for a single attribute linked to a single entity instance at some point in a problem state transition such as when there are multiple pegs available for a disc. UPRF defines an entity as an item having an imputable unique key that is associated to a set of attributes that vary in values between instances. Multiple entities result in the *entity state overflow* within the diagram that then generates the separate entities required for a particular problem instance. For example, in a three-disc Hanoi scenario, the simulator creates three entities. Figure 3-3 starts from the point of a particular instance after the entities creation and illustrates the *attribute overflow state* that occurs when there are multiple solution paths available to test. Each attribute overflow leads to a new branch since an attribute may only contain a single value within an entity instance within a problem instance.

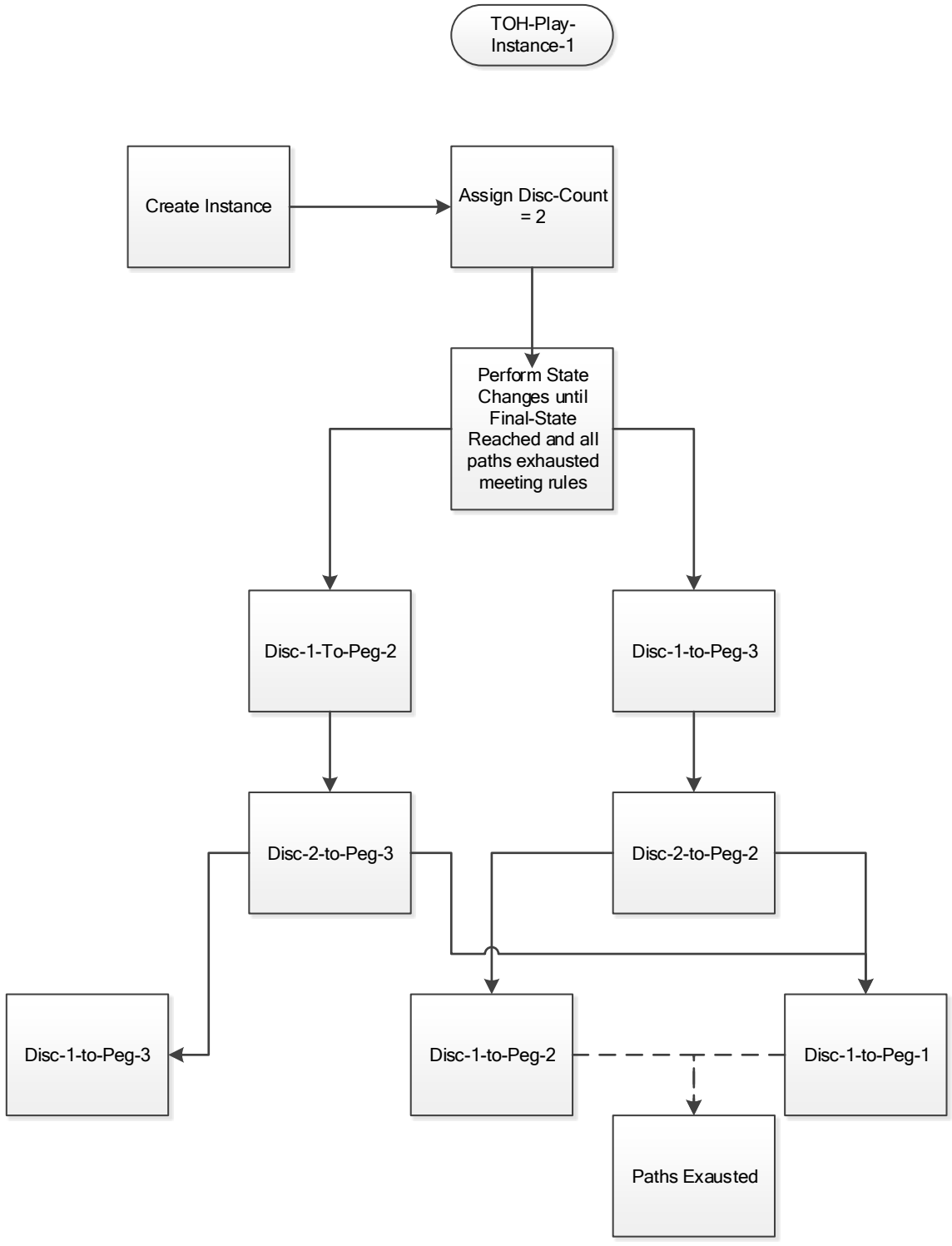


Figure 3-3: Attribute overflow in Hanoi Simulation

3.4 Data Architecture

UPRF is highly dependent on data and follows the case-based reasoning model alluded to in the literature review section outlining the generic schema approach. The framework records every operation endeavored to solve a problem in the database including the algorithms and functions used to carry out the operation and all of the parameters associated to the operation. Every step of the simulation, problem transformation, and problem solving process is traceable back to every attribute value modified. For the Tower of Hanoi, this means the framework captures every single move for every instance. The literature review for this dissertation was able to find only scant examples of systems with tracking at this level of detail. One reason for this may be practicality considerations associated with such massive data tracking requirements; however, newer technology is mitigating this.

Figure 3-4 illustrates the high-level data entities. The actual physical database design varies from this in order to achieve normalization. The relational design support representation of entities, attributes, and expressions that govern behavior or a problem solution endeavor as well as tracking of all operations that occur within the simulation and solving process including the transformation process described earlier.

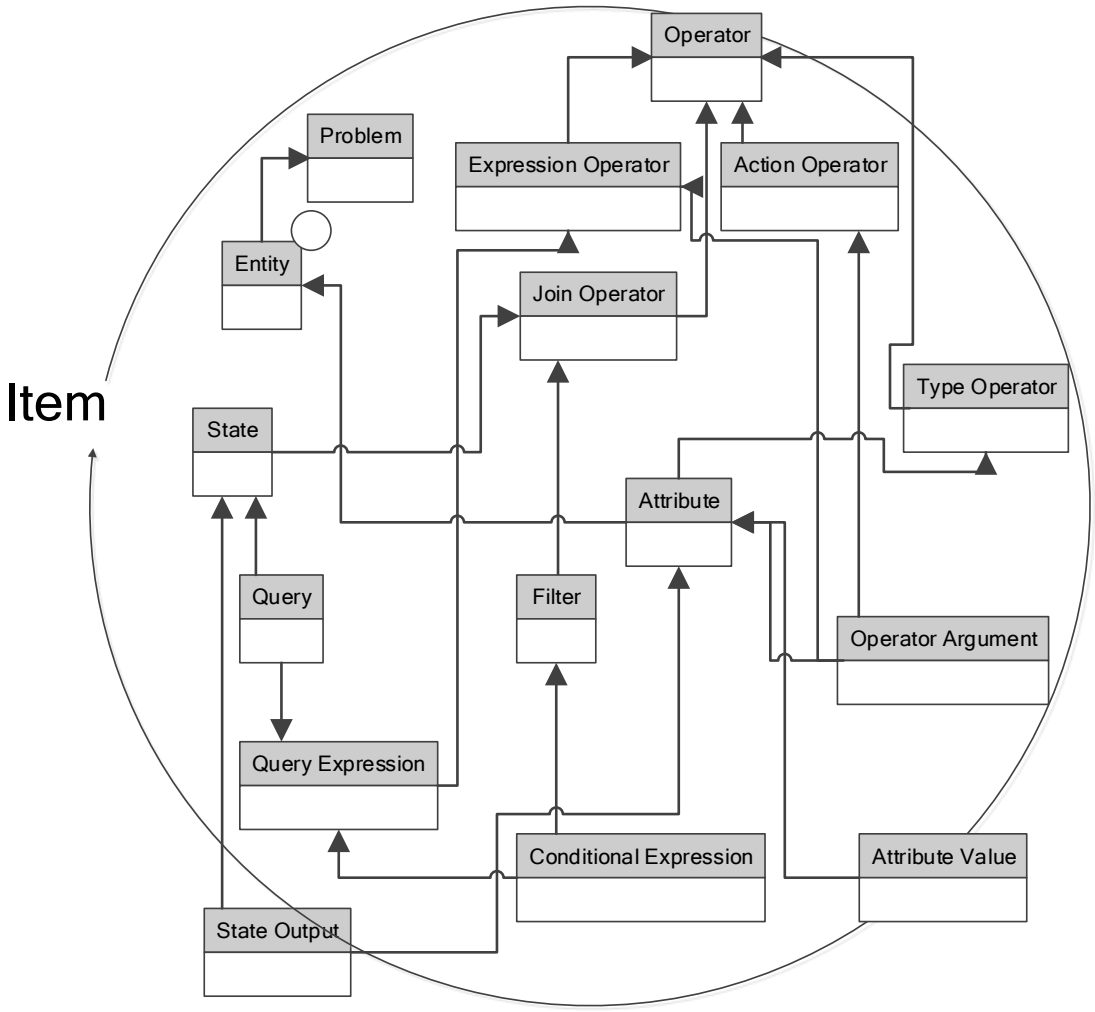


Figure 3-4: High Level Data Architecture

Data flows shown in Figure 3-5 show the outward flow from base level items. These items define constants or fixed attribute values through expressions transformed by operators. These operators then generate problem states that reflect the output of queries based on the allowed states. Ultimately, solution states are generated which capture the sequence of operations and the entities with their attributes and values that are affected throughout each step of a simulation process constrained by the state rules. The system endeavors to discover optimizations that generalize the

pursuit of solutions to higher order problems that return sequences of operations to perform transformations that generate solution paths avoiding the branching associated with brute-force simulation. For example, in the Tower of Hanoi, an optimization might be that the first move should always be to the second peg when the number of discs is even, but to the third peg when the number of discs is odd. In the operational proof, transform sequencing is utilized to identify patterns so that not only the correct starting move in Hanoi is calculated for any instance, but the entire solution pattern.

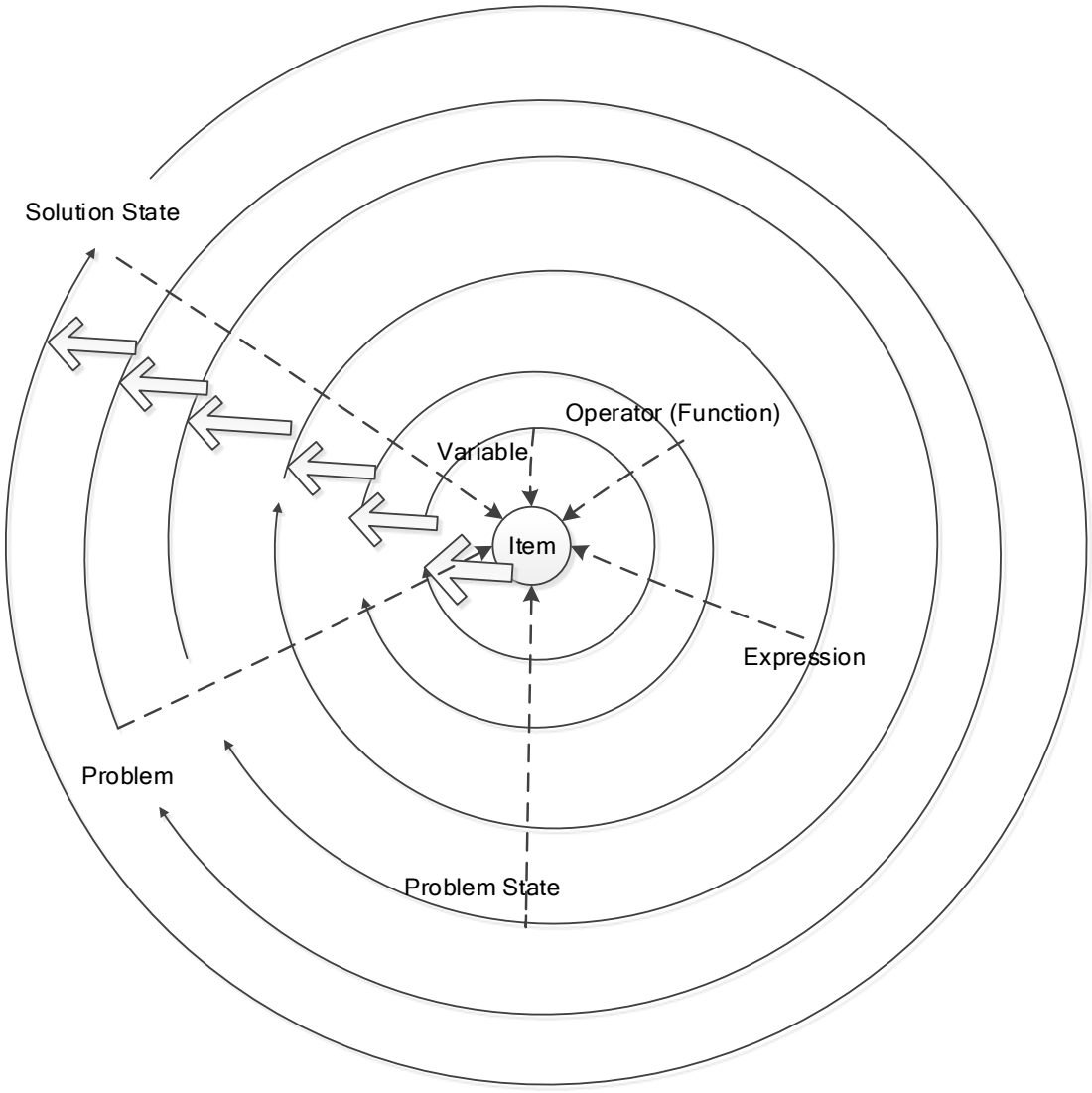


Figure 3-5: General Data Flow

3.5 Agent Topology

A common attribute of problem solving frameworks is an agent-based architecture. UPRF implements this type of architecture in the form of an agent topology. The idea of topology is that the agents themselves are independent and the framework easily supports the addition of agents. Stateless data architecture and task-oriented processing promote the use of independent processes

that may be distributed. UPRF's agent topology framework meets the goals outlined by the literature for agent-based modelling and simulation [116] which include the following:

- An agent is identifiable with discrete sets of characteristics and self-contained.
- An agent lives in an environment that it interacts with other agent with the capability to respond to the environment.
- An agent is goal-directed having goals to achieve with respect to its behaviors
- An agent is autonomous and self-directed.
- An agent is flexible, and has the ability to learn and adapt its behaviors over time based on experience. An agent may have rules that modify its rules of behavior.

Multiple agents collaborate to support problem definition, simulation, and higher order transformations. Figure 3-6 show the main agents and their interactions with a broker agent that dispatches tasks. The design model follows that of a "smart environment" [117] which promotes an environment where the behaviors of each object assist each other to reach the goals of the framework. This design provides scalability across parallel processes across any number of machines. Chapter 5 discusses some of the deployment options that UPRF can leverage to take advantage of distributed high-performance computing frameworks. The agent architecture is a very important concept in intelligent systems. Frameworks that can seamlessly integrate independent agents that focus on solving a specific task provides an environment that heuristics can be seamlessly added, which enhance the other agents. Lifelong machine learning systems provide a model of this approach wherein a heuristics generator operates independently and autonomously against the problem stream [118]. This behavior occurs in UPRF through separation

of the various tasks for assimilation of knowledge from those of simulation and transformation of problems.

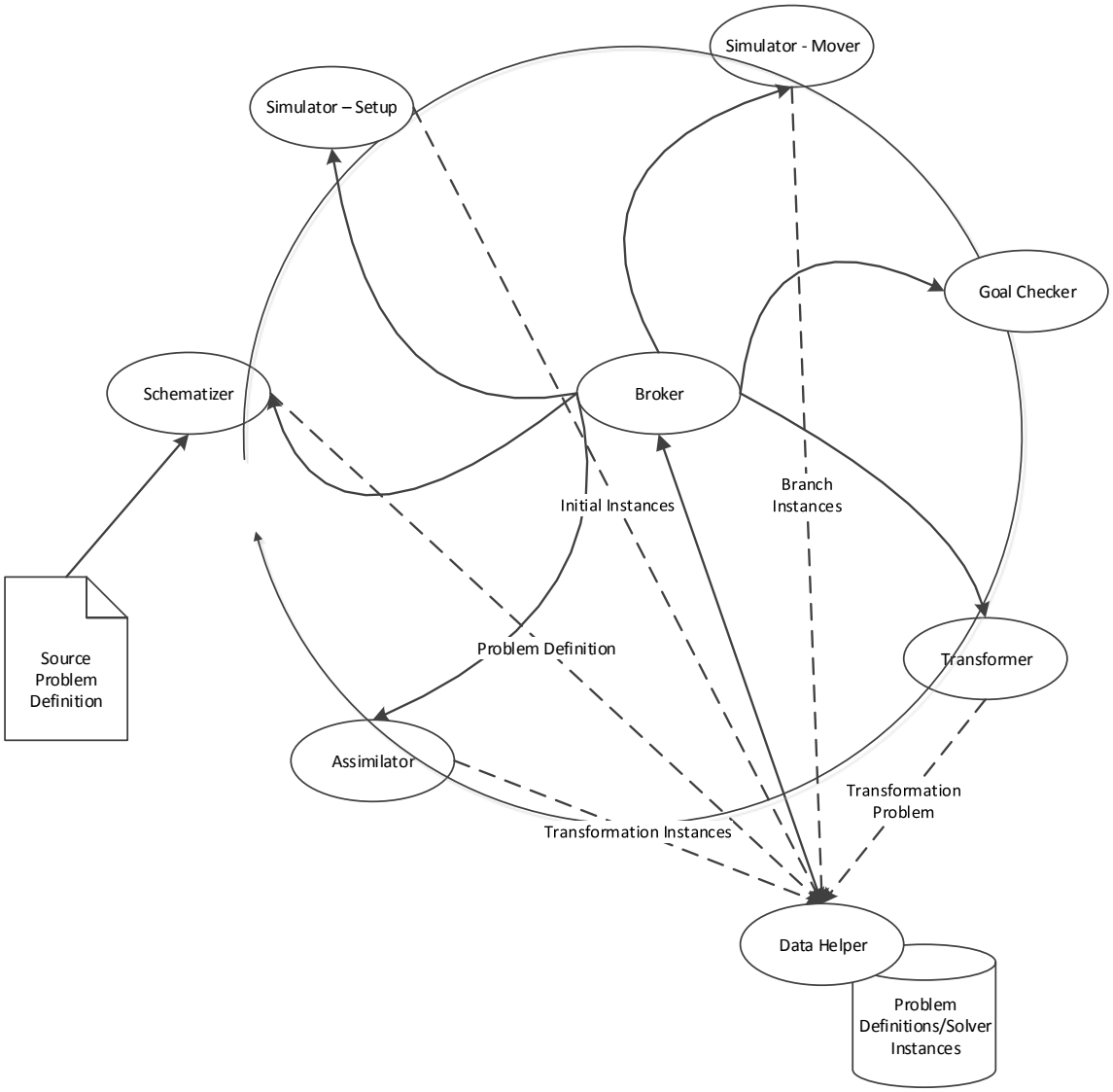


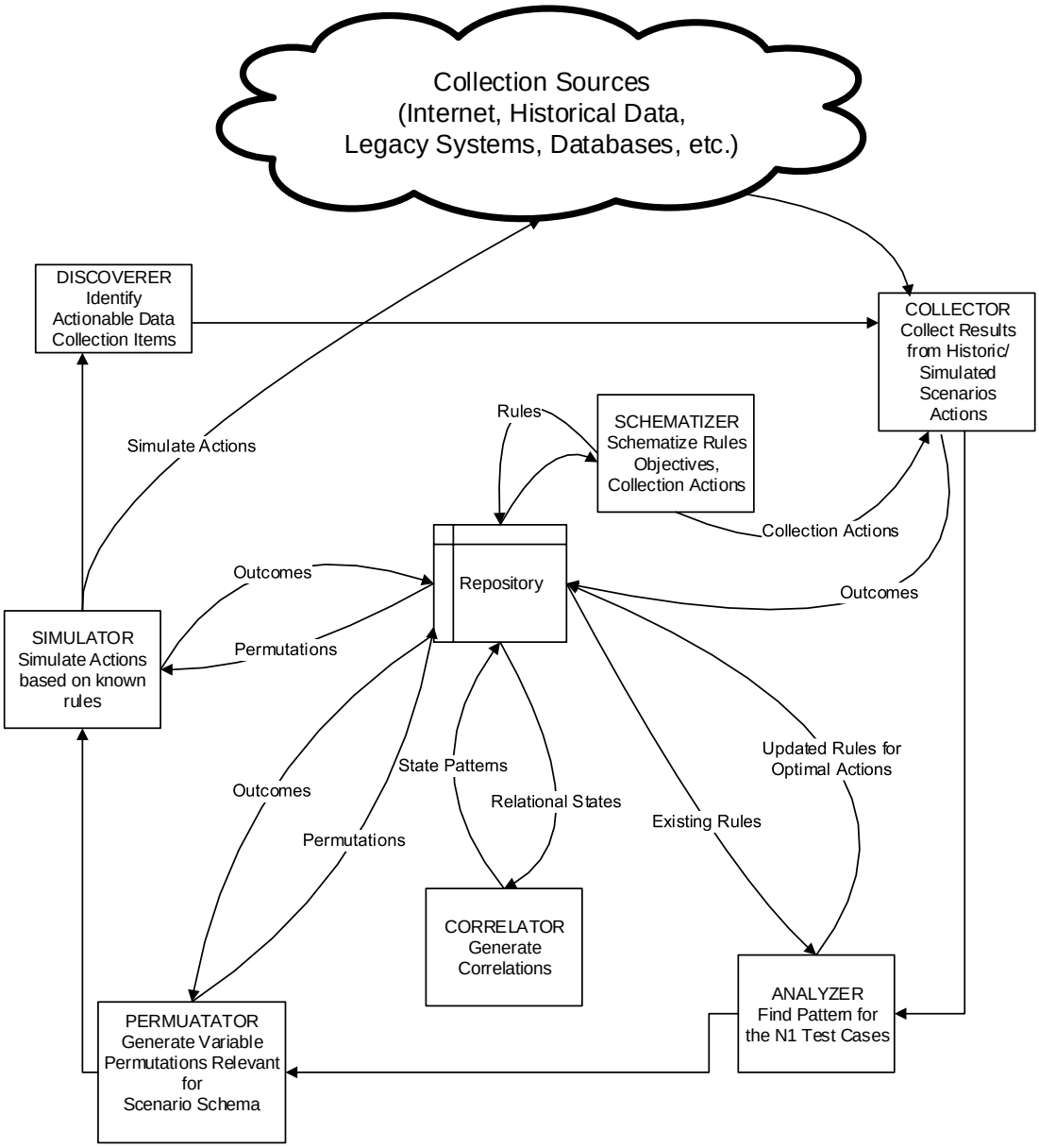
Figure 3-6: Agent Architecture

Figure 3-6 illustrates the relationships between the major agents with their roles listed below:

- Schematizer: Listens for problem definitions and injects into the database

- Simulator-Setup: Instantiates the starting instances for a problem. For example, this allows dispatching of tasks to create instances for disc count of two to ten given these as the minimum and maximum number of discs.
- Simulator-Mover: Performs depth/breadth exploration of simulation solution space based on the transition query for the problem for instances in the initial or checked-state still requiring transitions.
- Goal-Checker: Checks the simulation instance against the goal for instances that have just completed a transition state.
- Transformer: Checks for existence of combinations of at least two solved instances to escalate to the next higher order problem to identify the transformation sequence needed to derive a solved instance from another solved instance within a problem space.
- Assimilator: Generates transformation problem instances for the transformer problem.
- Broker: Looks for tasks in the database that are relevant to the particular agents and dispatches the tasks to the agent.
- Data Helper: Provides the data interface layer between the broker and various agents to the problem definition and problem instances.

The architecture supports knowledge discovery with a feedback loop. In Figure 3-3, a web discovery agent integrates into the framework to engage a continuous improvement model that integrates semantic web search with unstructured text mining. Potential interactions depicted in Figure 3-7 as part of an overall learning framework.



NOTE: A Relational State is the matrix of outcomes correlated to actions for a specific Entity. State Patterns are Relational States Correlated to Relational States for other Entities including the Time entity as well as other State Patterns.

Figure 3-7: Automation Framework

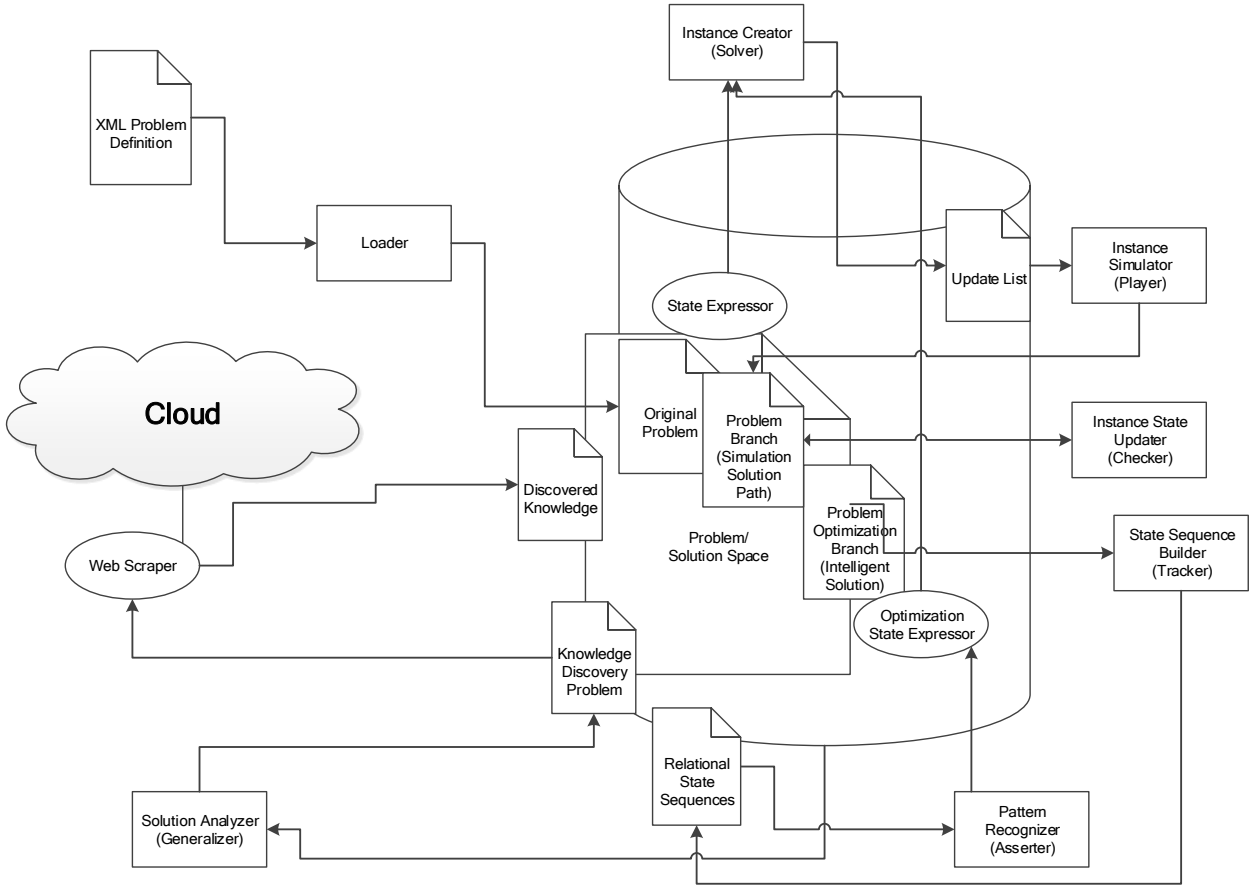


Figure 3-8: Web Mining Scenario with Feedback

3.6 Problem Schematization

UPRF includes a universal problem definition language (UPDL) component. UPDL provides an XML format that works with a listener process that imports problem definitions into the system as illustrated in Figure 3-8. Figure 3-9 depicts the UPDL schema. Figure 3-10 shows an example with Tower of Hanoi that adheres to the schema. The schema supports defining problems in a structure that is compatible with the core relational database storage design. During the course of

the dissertation, the schema has evolved. Utilizing a technique based on normalizing OML storage in relations [119], a mapping layer provides transformation from the schema to the actual database. The schema is thus extensible and changeable to better accommodate problem definitions. Therefore, UPRF may evolve to support additional schemas focused on particular problem domains that may provide greater abstraction on top of the core schema or support lower level functions and external data interfaces. The mapping layer ensures that different schemas can still populate the core database entities defined in the data architecture section.

Figure 3-9: UPDL Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<Upr xmlns:xsi="ProblemDef-v12.xsd" Version="11.0">
  - <Problem TransitionQuery="Play-Game" FailQuery="Check-if-Lost" GoalQuery="Check-If-Won"
    DeriveFrom="Disc-Count" Id="Hanoi">
      - <Attribute Id="Start-Disc">
            <Value>1</Value>
        </Attribute>
      - <Attribute Id="Min-Disc-Count">
            <Value>1</Value>
        </Attribute>
      - <Attribute Id="Max-Disc-Count">
            <Value>10</Value>
        </Attribute>
      - <Attribute Id="Start-Peg">
            <Value>1</Value>
        </Attribute>
      - <Attribute Id="Final-Peg">
            <Value>3</Value>
        </Attribute>
      - <Attribute Id="Step-Count">
            <Expression>Get-Step-Count</Expression>
        </Attribute>
      - <Attribute Id="Disc-Count">
            <Expression>Disc-Count-Range</Expression>
        </Attribute>
      - <Attribute Id="Peg-List">
            <ValueList>1,2,3</ValueList>
        </Attribute>
      - <Attribute Id="Disc-Size">
            <Expression>Disc-Size-Range</Expression>
        </Attribute>
      - <DataEntity DeriveFrom="Disc-Size" Id="Disc">
            <Attribute Id="Peg" Domain="Peg-List" Default="Start-Peg"/>
        </DataEntity>
            <!-- All attributes, problems, and entities are automatically generated as expressions and available
            for use as expressions in the database. Attributes based on constants are generated as expressions
            as needed -->
      - <Expression Id="Max-Move-Count" Step="0" Operator="POWER">
            <Argument Parameter="Exponent" Attribute="Disc-Count"/>
            <Argument Parameter="Base" Attribute="2"/>
        </Expression>
      - <Expression Id="Min-Disc-For-Peg" Step="0" Operator="MIN">
            <Argument Attribute="Disc-Size"/>
        </Expression>
      - <Expression Id="Disc-Count-On-Peg" Step="0" Operator="COUNT">
            <Argument Attribute="Disc-Size"/>
        </Expression>
      - <Expression Id="Candidate-Disc" Step="0" Operator="SELECT_NONUSED">
            <Argument Attribute="Disc"/>
        </Expression>
      - <Expression Id="Disc-Peg" Step="0" Operator="SELECT">
            <Argument Attribute="Peg"/>
        </Expression>
      - <Expression Id="Disc-Count-Range" Step="1" Operator="GENERATE_RANGE">
            <Argument Parameter="From" Attribute="2"/>
            <Argument Parameter="By" Attribute="1"/>
            <Argument Parameter="To" Attribute="Max-Disc-Count"/>
        </Expression>
```

```
- <Expression Id="Disc-Count-Range" Step="1" Operator="GENERATE_RANGE">
      <Argument Parameter="From" Attribute="2"/>
      <Argument Parameter="By" Attribute="1"/>
      <Argument Parameter="To" Attribute="Max-Disc-Count"/>
  </Expression>
- <Expression Id="Disc-Size-Range" Step="1" Operator="GENERATE_RANGE">
      <Argument Parameter="From" Attribute="1"/>
      <Argument Parameter="By" Attribute="1"/>
      <Argument Parameter="To" Attribute="Disc-Count"/>
  </Expression>
- <Query Id="Play-Game">
      <Extract Id="Next-Disc" OutputAttribute="Disc" Expression="Candidate-Disc"/>
    - <Extract Id="Disc-Peg" OutputAttribute="Peg" Expression="Disc-Peg">
          <Criteria Attribute="Disc" CompareOperator="EQUAL" Extract="Next-Disc"/>
              <!-- Get the disc for each peg -->
      </Extract>
    - <Extract Id="Next-Peg" OutputAttribute="Peg" Expression="Peg-List">
          <Criteria Attribute="Peg-List" CompareOperator="NOT_EQUAL" Extract="Disc-Peg"/>
              <!-- Expand to include peg list except for the current disc for next peg-->
      </Extract>
    - <Extract Id="Min-Disc-For-Disc-Peg" Expression="Min-Disc-For-Peg">
          <Criteria Attribute="Peg" CompareOperator="EQUAL" Extract="Disc-Peg"/>
              <!-- Find out what the smallest disc is for the peg of each disc-->
      </Extract>
    - <Extract Id="Min-Disc-For-Next-Peg" Expression="Min-Disc-For-Peg">
          <Criteria Attribute="Peg" CompareOperator="EQUAL" Extract="Next-Peg"/>
              <!-- Find out what the smallest disc is for the candidate pegs for each disc Use NullExpression
              to set value to disc-count for peg that has no min. This attribute effectuates an optional join
              to the expression as well as providing the placeholder value-->
      </Extract>
          <!-- We now have all information needed in the matrix to filter for next move: Example: Assume
          disc 1 on peg 2 and disc 2 and 3 on peg 1 with disc 1 moved last. Disc 1 is eliminated since it was
          moved last, Pegs for each disc eliminated in lookup that filters using NOT_EQUAL current disc peg
          Candidate-Disc Disc-Peg Next-Peg Min-Disc-For-Peg Min-Disc-For-Next-Peg 2 1 2 2 1 2 1 3 2 3* 3
          1 2 2 1 3 1 3 2 3* * Instance Disc-count applied due to use of NullExpression option -->
      <Filter Id="Filter-Disc-Too-Large-For-Peg" Operator="LESS_THAN" Extract2="Min-Disc-For-
          Peg" Extract1="Candidate-Disc"> </Filter>
          <!-- This eliminates all but row 2 above indicating the only valid next move is Disc 2 to Peg 3-->
  </Query>
- <Query Id="Check-If-Lost">
      <Extract Id="Max-Move-Count" OutputAttribute="1" Expression="Max-Move-Count"/>
          <!-- Verify that total number of moves is less than 2 raised to the number of discs -->
      <Extract Id="Step-Count" Expression="Step-Count"/>
      <Filter Id="Check-Move-Count" Operator="EQUAL" Extract2="Max-Move-Count"
          Extract1="Step-Count"/>
  </Query>
- <Query Id="Check-If-Won">
      <Extract Id="Disc-Count" OutputAttribute="1" Expression="Disc-Count"/>
          <!-- Verify that all discs are on Peg 3-->
      <Extract Id="Peg3" Expression="Final-Peg"/>
    - <Extract Id="Disc-Count-Peg3" Expression="Disc-Count-On-Peg">
          <Criteria Attribute="Disc" CompareOperator="EQUAL" Extract="Peg3"/>
      </Extract>
  </Query>
</Problem>
/Upr>
```

Figure 3-10: Sample Problem Definition - Tower of Hanoi

Figure 3-10 shows the Tower of Hanoi in UPDL format with entities, attributes, expressions, and queries to govern the state. This provides the information needed for the framework to define

the problem so that a simulation process can explore the valid solutions. The next chapter containing the operational proof explains how the items interact to provide queries that associate to states and determine whether a simulation instance completes successfully or fails.

UPRF provides an import process for the problems written according to the UPDL specification. This process loads the problems into a staging area and then normalizes the data from the schema into the database. The relational database design also provides support for historical tracking of imports and lineage to higher order problem transformations. Figures 3-10 and 3-11 show the process for importing problems written in UPDL format into UPRF.

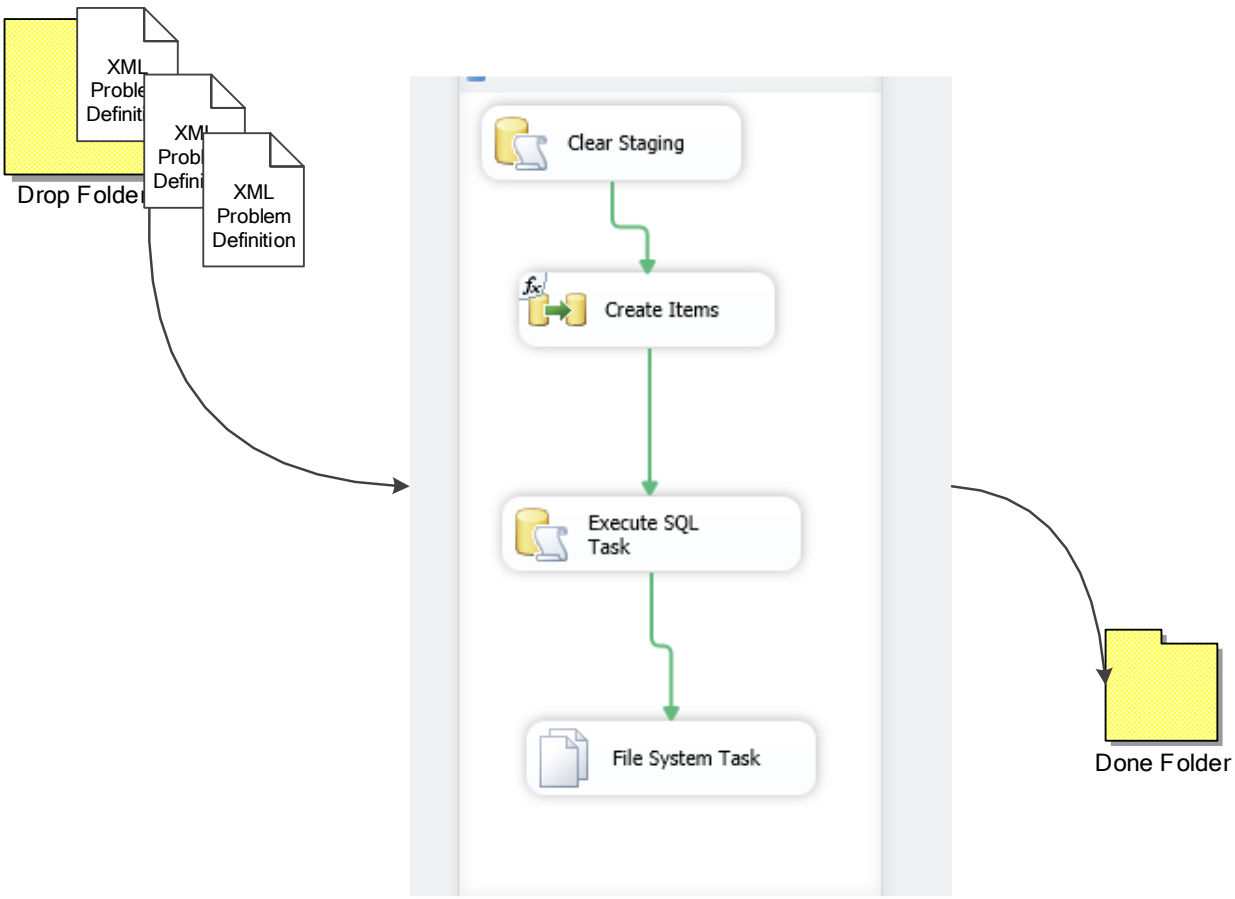


Figure 3-11: High-level Problem Load Process

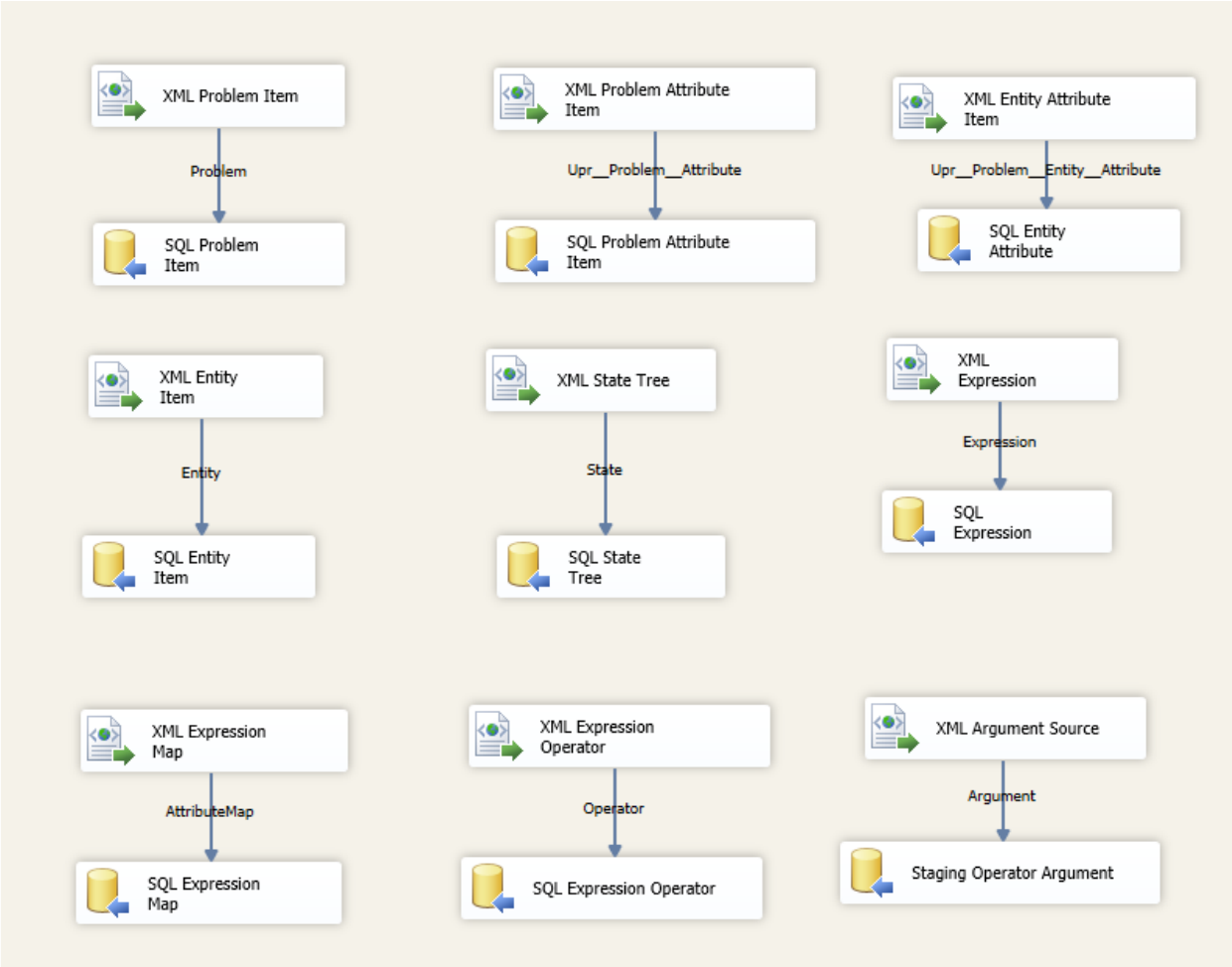


Figure 3-12: Detailed UPDL Flow into Database

3.7 Database Implementation

The relational database consists of four main schemas:

1. Staging: Defines the area for loading UPDL problems from XML. This area receives problems schematized from the UPDL XML node structure that the framework transforms into the Data schema after validation finishes (Figure 3-12).
2. Data: Defines the problem (Figure 3-13).

3. Engine: Tracks all entities and attribute values over a solution instance (Figure 3-14).
4. Reference: Defines the operators (functions) utilizable for all operation along with their parameters (Figure 3-15). Expressions in the Data schema utilize operators in the Reference schema for problem instances, which enables tracking all operator usages correlated to problem instances.

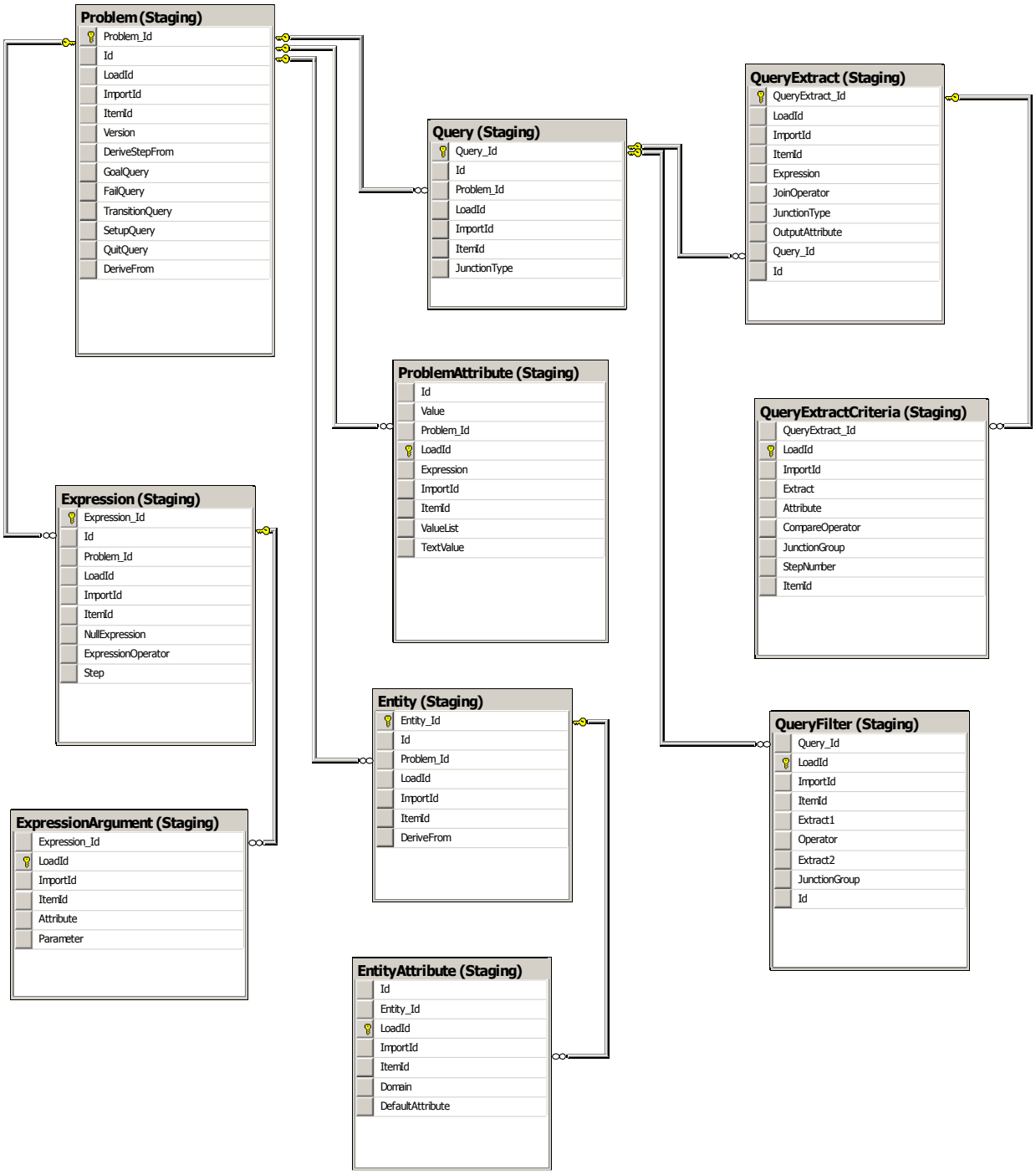


Figure 3-13: Staging Schema

The staging schema stores problem definitions that come through the UPDL loading process and potentially support other import formats as well. This schema allows verifying the integrity of problem schemas before presentation for solving in UPRF. Staging provides the area to persist the XML definition and then validate the definition before transforming it into the Data schema.

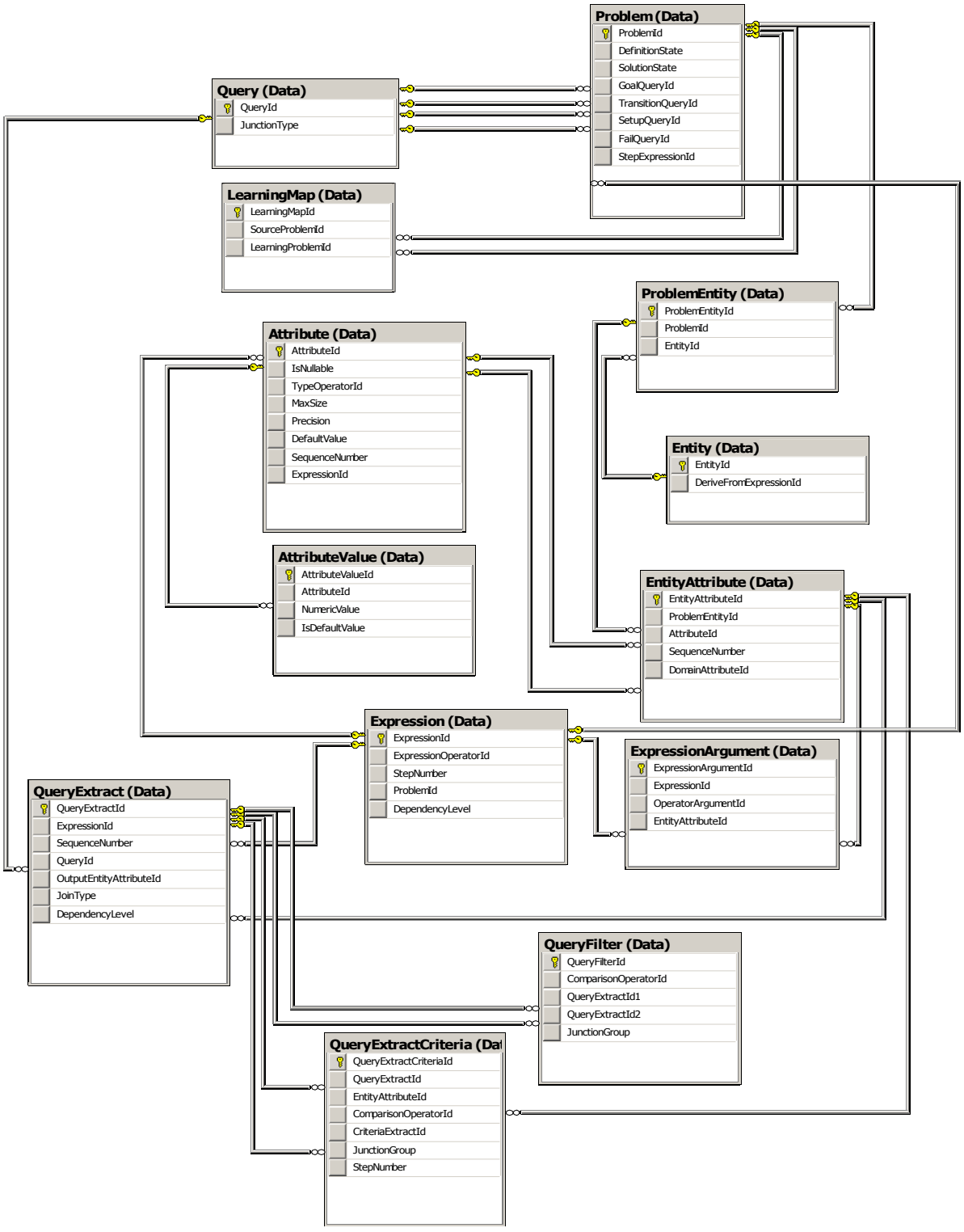


Figure 3-14: Data Schema

The Data schema defines the database objects required to model a problem definition including entities, attributes, expressions, and queries to represent the initial state, valid transition states, failure/abort states, and goal state for a problem. It also provides a mapping (LearningMap table) for linking instances of one problem to a learning (transformation) problem for higher order generalization of the simulation instance solutions. It correlates loosely to the schema defined by UPDL and normalizes the incoming problem definition.

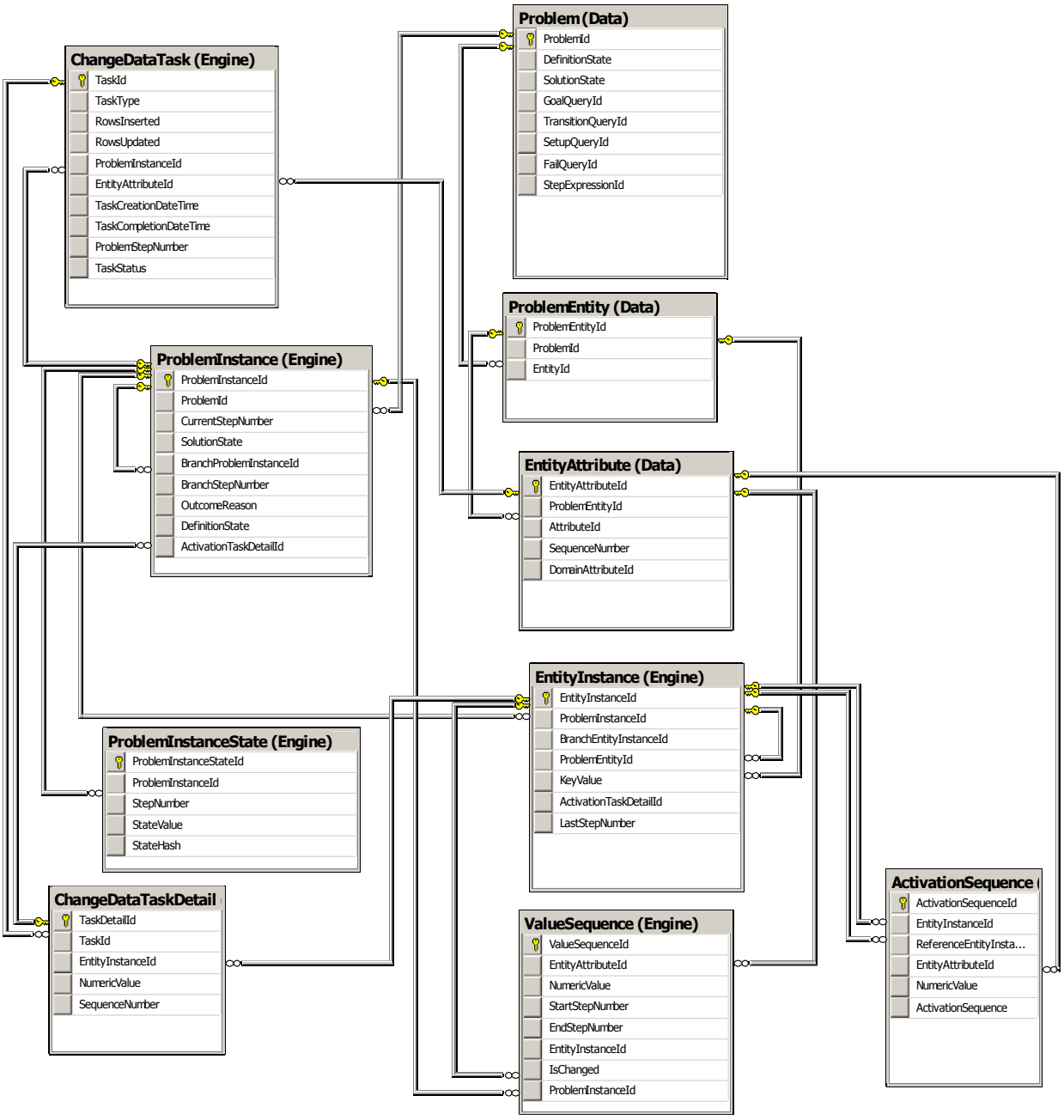


Figure 3-15: Engine Schema

The Engine schema stores all of the instances of a problem so that the simulator can carry out solution attempts. It includes a mapping mechanism for parallelizing tasks to carry out via the ChangeDataTask and ChangeDataTaskDetail. This schema integrates with the problem definition

schema to enable a problem to spawn solution instances. This mechanism supports distributed and parallel execution of solution instantiations at the instance state level. The Engine schema also stores the relational state sequences associated with all attribute value changes including the state activations for all recorded values for an attribute over a solution state sequence.

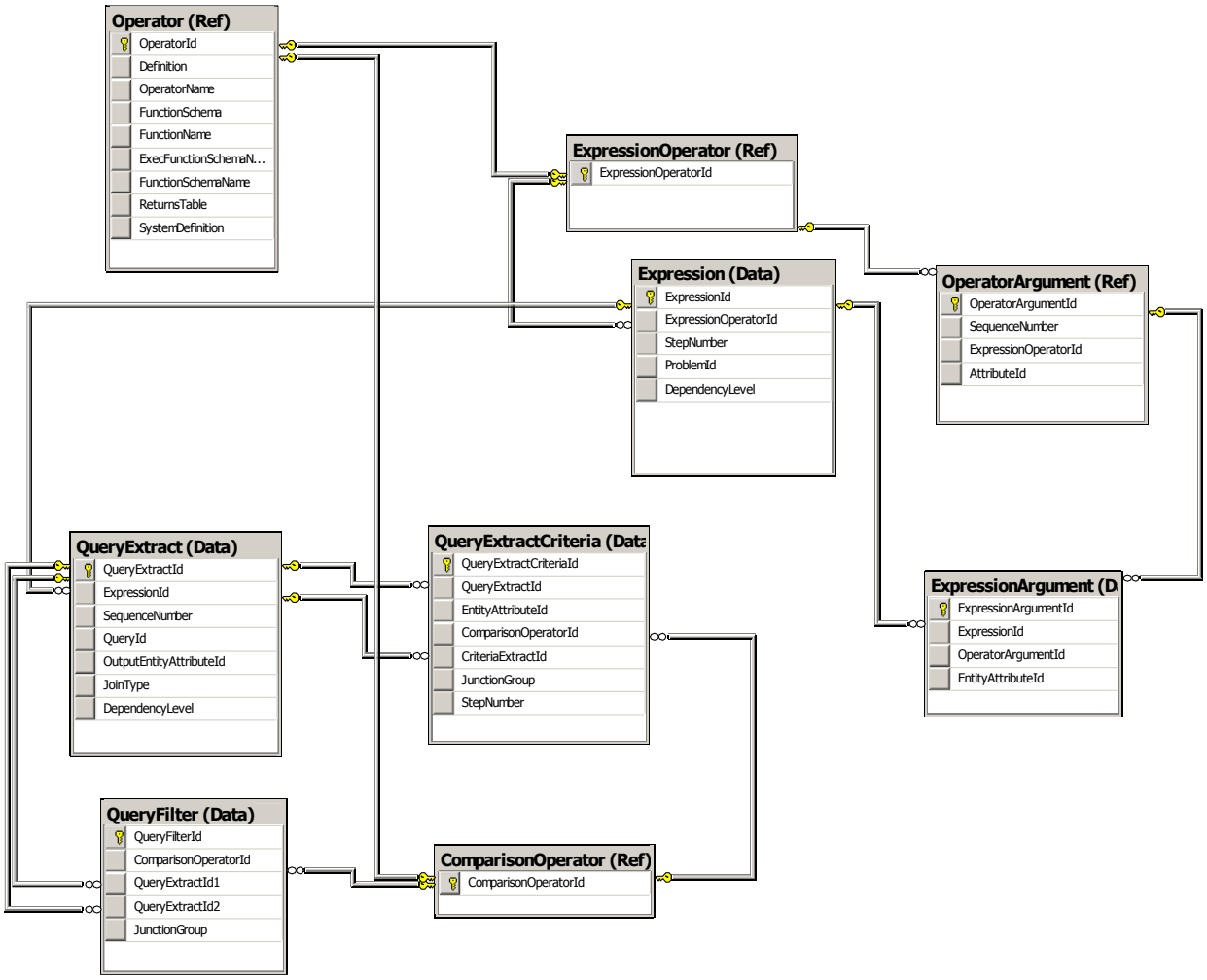


Figure 3-16: Ref Schema

The Reference (Ref) schema supports extensibility for all types of operators (functions) utilized for simulation and solution discovery. It defines the operators that are available to the system and loads operators dynamically. This provides extensibility to add new operators to the

system by simply importing the function into the engine schema and then referencing the function from the problem definition.

Operators are enumerable based on type and then executed in a higher order simulation to search for the operators that can generate a solution path for one instance of a problem using data from another instance of a problem. In this way, the framework tracks the sequences of selected operators correlated to a solution path associated to a problem instance. These operators include conditional operators to compare values as well as expression operators that can generate scalar or tabular result sets. Expression operators include transformation operators that can host machine-learning algorithms in order to generate prediction sequences from source sequences.

3.8 Simulation

By default, the simulator utilizes depth-first search to explore solution spaces for each problem's instances. Each problem instance is independent allowing UPRF to explore the solution space separately and branch instances to generate new instances. The new instances branch off in the same fashion as the originating instances for parallel execution. Problem queries that define the candidate result sets drive the exploration of the solution space. The search is not strictly depth-first if the problem itself exposes queries that lead to a different type of search including breadth. As the search proceeds, the framework catalogues the candidate solutions. Simulation instances terminate upon reaching a certain resource threshold if a "quit query" specifies to stop execution. (Figure 3-1) shows the process for assigning tasks through state inspection as illustrated in Figure 3-17.

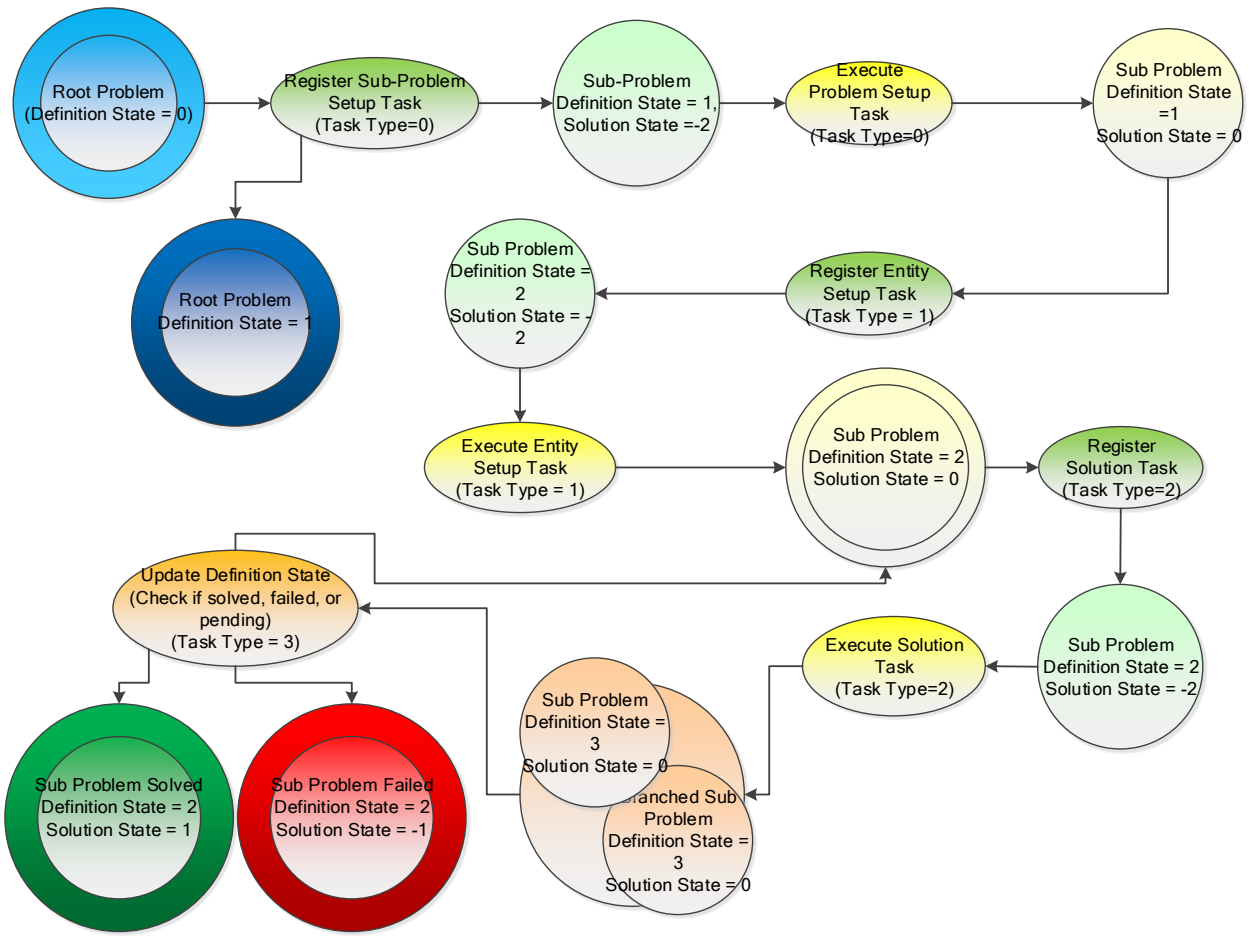


Figure 3-17: Simulation State Flow

Figure 3-17 shows the problem-instance state flow in terms of color shading to indicate transition for execution by various processes. When a problem instance enters the light-green type of state, this indicates it is ready for some sort of execution. The dark green tasks register instances for the next task while the yellow tasks execute. The end state shown by the double-circle green object indicates termination due to success in reaching the goal state while the red state indicates termination due to failure. As each task in the cycle is executed, the definition and/or solution state (depending upon the task) is updated which then qualifies the next task to be executed for the

problem instance. Figure 3-18 illustrates the creation of new states for a problem based on results from queries and Figure 3-19 depicts the query processing that generates potential states.

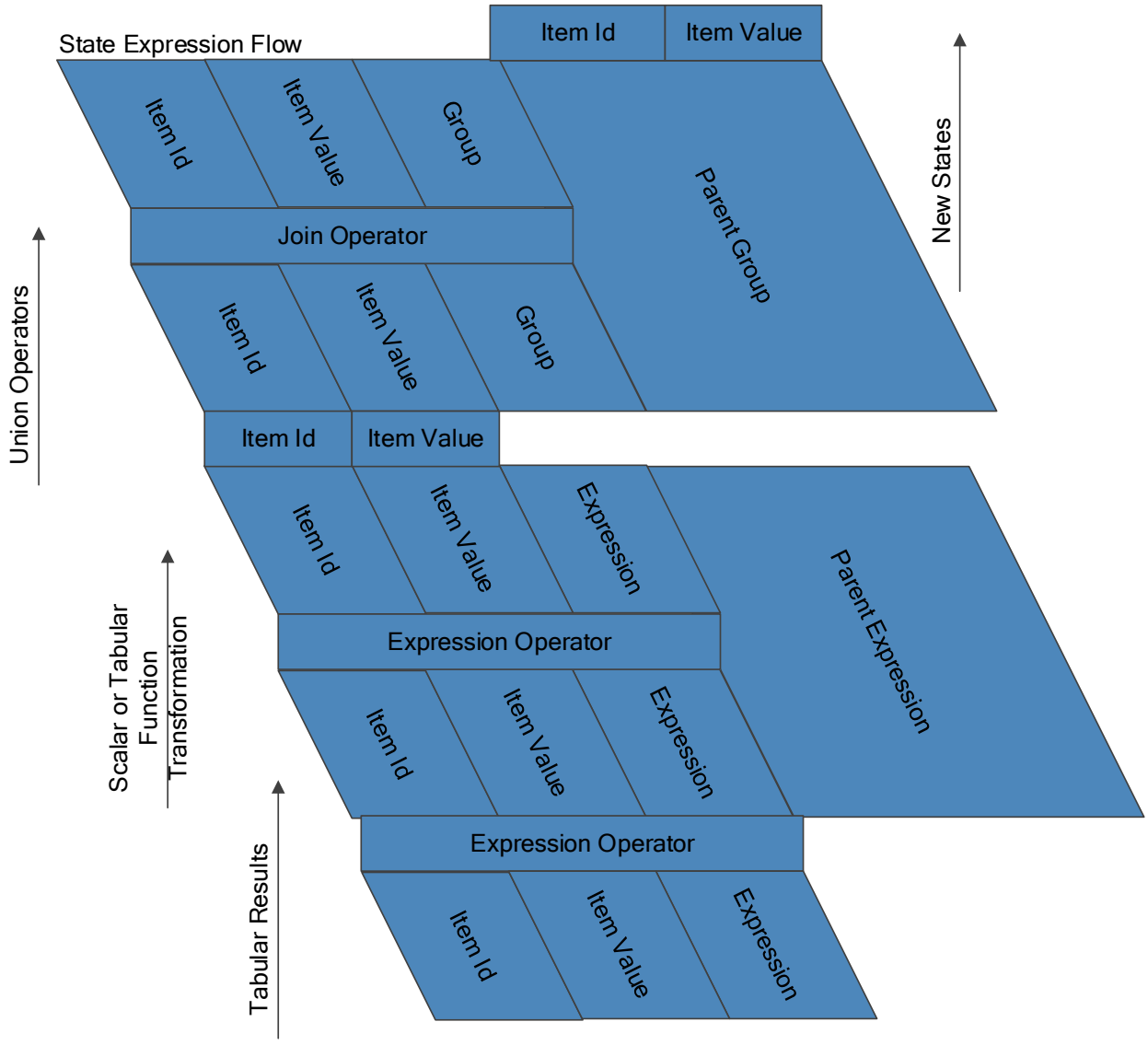


Figure 3-18: State Expression Flow

Expression operators interact with entities and attributes to return results through query extractions. Query extractions are joined together to form results and then filtered. UPRF spawns new states when more than one result returns from a transition query and creates an overflow condition. The query-processing engine generates results through the extraction and filters with logical conjunctions that represent Boolean and/or groupings (Figure 3-19).

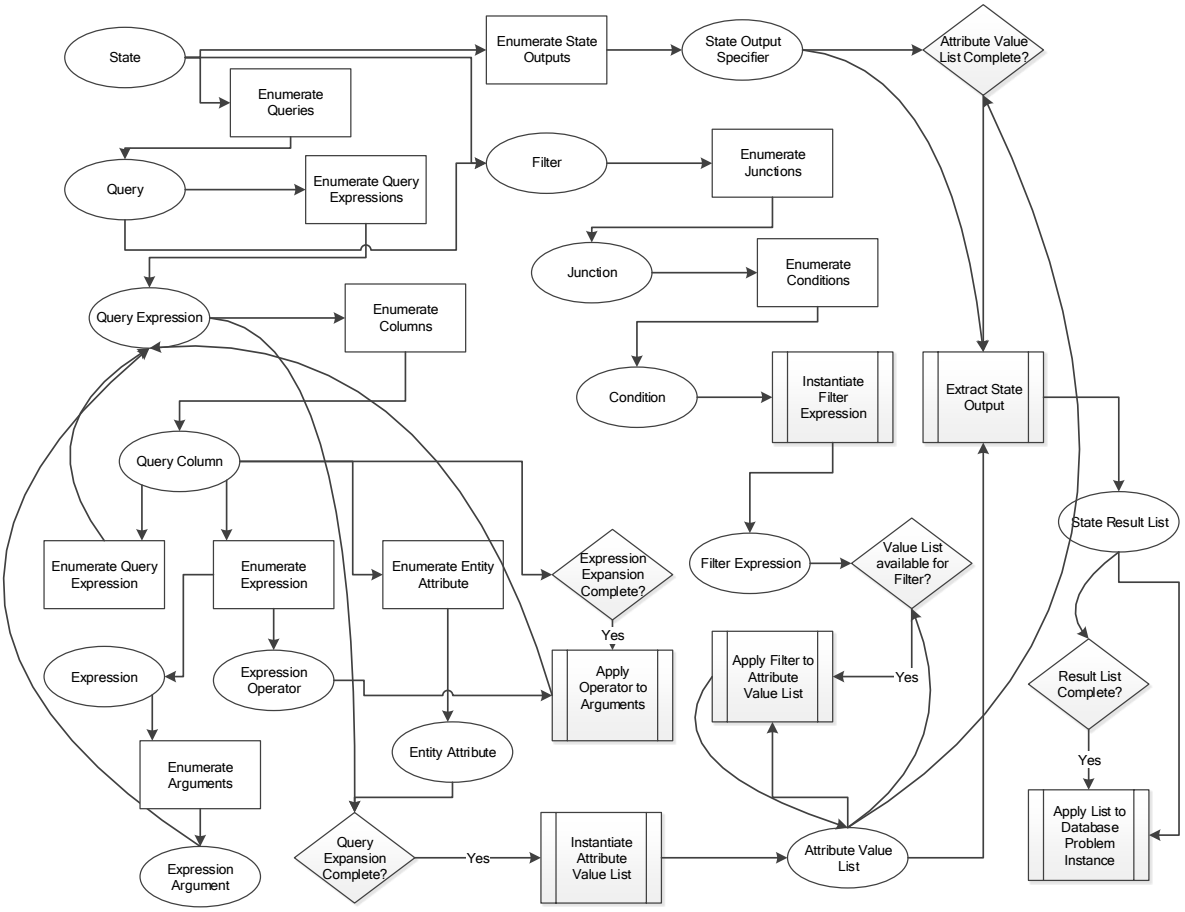


Figure 3-19: State Generation from Queries

3.9 Transformation

UPRF performs transformation automatically upon solution sequence completion upon solution path discovery for a second instance of a problem. The transformation results in a new

instance of the Universal Discovery Problem (UDP) (elaborated on in chapter 4). The UDP defines the goal state as the sequence of operators required to transform one instance to another instance. For example in the Tower of Hanoi, when a four-disc problem instance is solved after a three-disc instance has also been solved, then a transformation problem is generated whose goal is to generate the operations required to utilize solution sequence paths from the three-disc simulation to generate the solution for the four-disc instance. Chapter 5 delves into the 4-peg Hanoi or variable-peg (K-peg) scenario to show how UPRF can scale problem learning a simple base case to more advanced solutions. Figure 3-20 depicts the flow for transforming simulations into higher order problems.

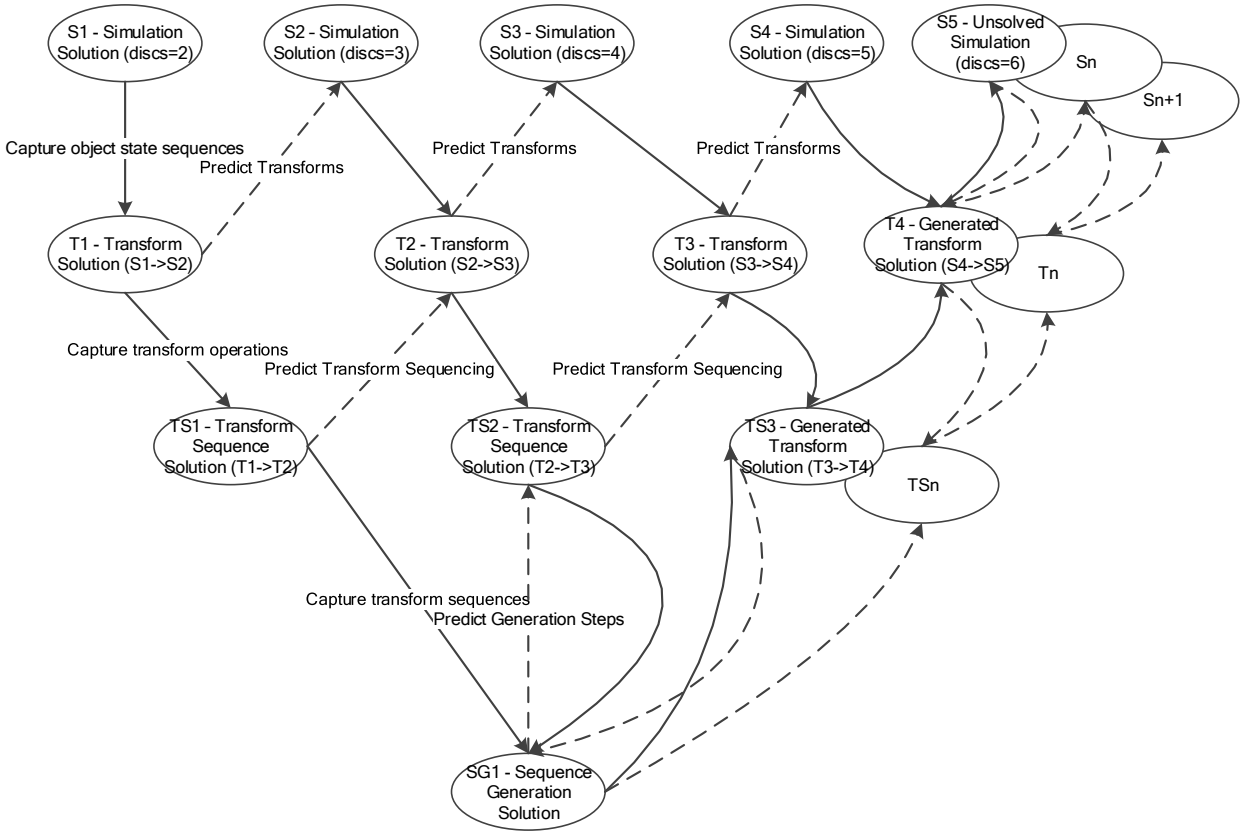


Figure 3-20: Problem Transformation Process

3.10 Chapter Summary

This chapter provided a high-level software architecture for a universal problem resolution framework (UPRF). It did not delve into implementation details, but did provide a general design for critical components. Chapter 4 builds upon this architectural foundation and utilizes these components in order to demonstrate an operational proof. Chapter 3 outlined the Tower of Hanoi problem as the base use case for UPRF validation and established key constructs for UPRF architecture, which include:

1) A core architecture that provides a continuous improvement cycle to support transforming learning achieved at a lower level to increasingly higher levels in the search for a general solution or problem optimization;
2) A data architecture that supports generic representation of problem information including the various states of a problem from instantiation through solution;
3) An agent topology that shows the integration of various agents working together within the framework to enable continuous improvement;

In addition, the chapter outlined the functionality for problem schematization, simulation, and transformation and delved into the actual database implementation undergirding the framework.

4. OPERATIONAL PROOF

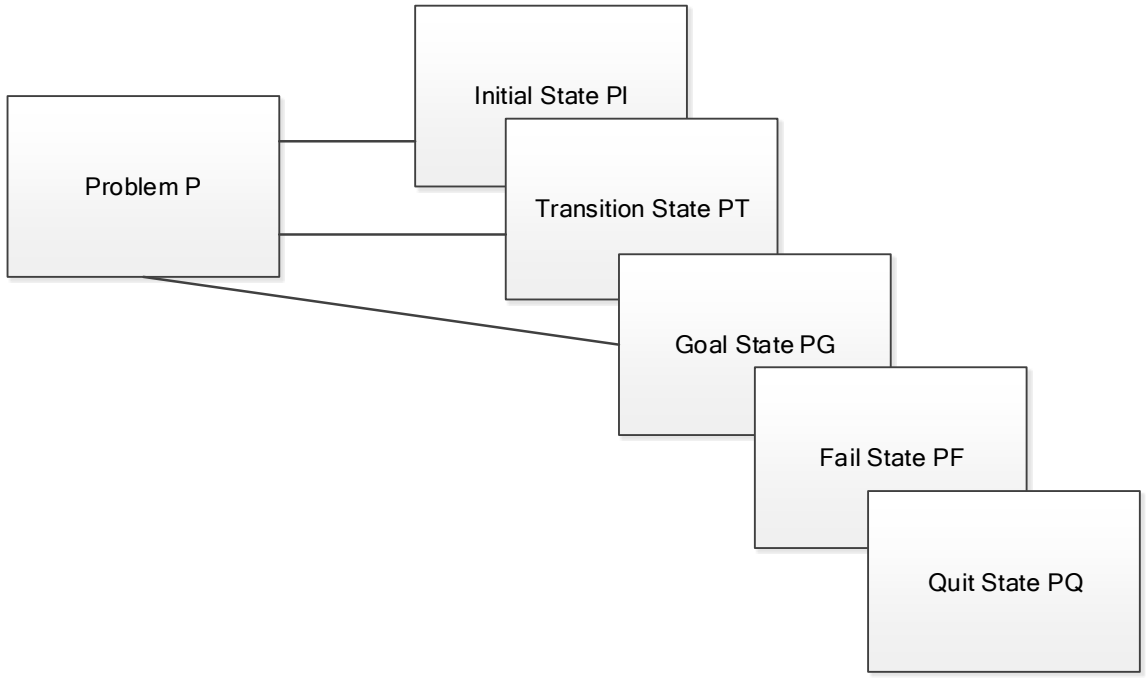
4.1 Inductive Proof for Feasibility

A primary goal of this work is to construct a framework that utilizes scenarios of adequate variation and sufficient complexity such that it achieves the goal of continuous improvement. To validate the following proof, the dissertation provides scenarios that demonstrate how to schematize problems and then pursue solution paths through simulation, which result in higher-order transformation problems. The transformation problem fits within the same schema as the underlying problem, enabling a simulative process that focuses on optimization of solution paths at increasingly higher order degrees, which can target and benefit from the entire breadth of problems presented to the framework.

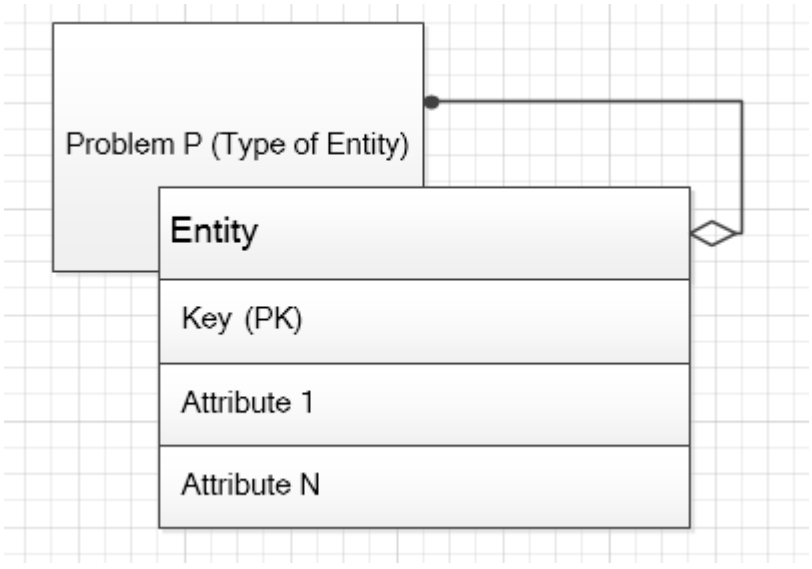
4.1.1 Generic Problem Definition and Simulation Capability

4.1.1.1 Postulates

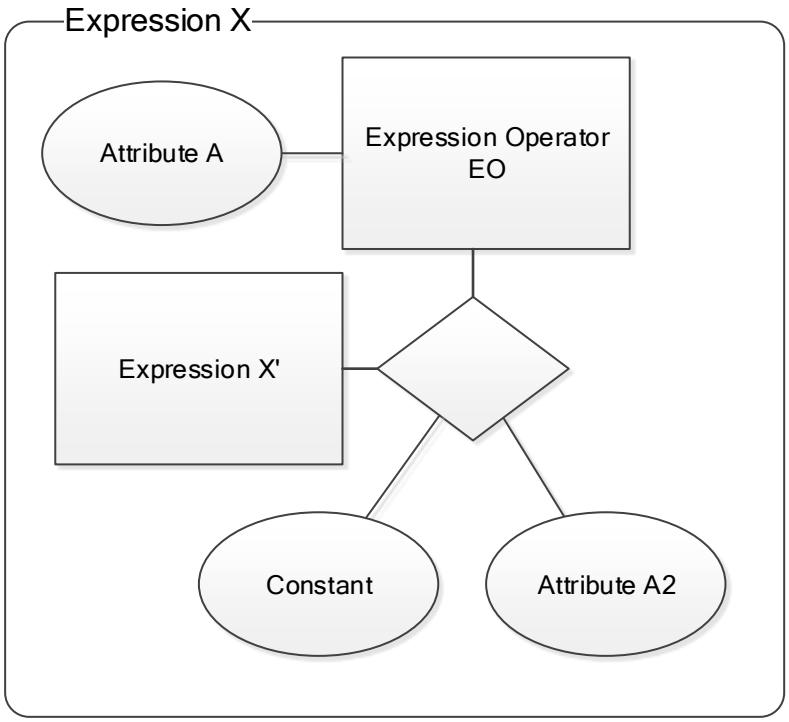
1. Let P be a Problem having initial state PI, allowed transition states PT, and goal state PG and optional state for failure (PF) and quitting (PQ)



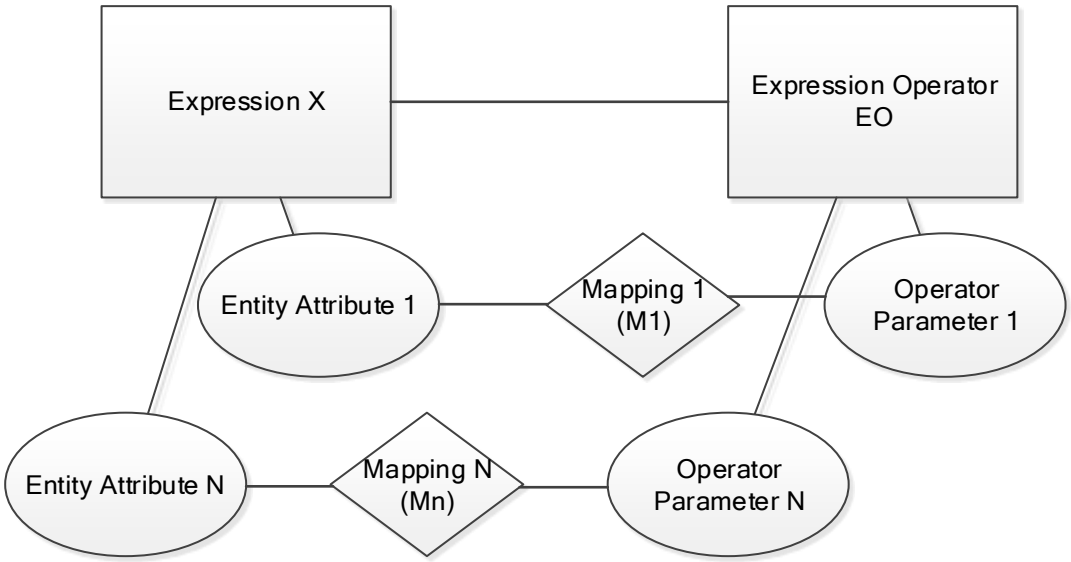
2. Define P in terms of one or more entities E including itself having one or more attributes A. Represent entity E by a key attribute K that uniquely identifies it and does not change in value. Let an entity E map to an attribute A as EA such that EA derives from the domain or expression of attribute A.



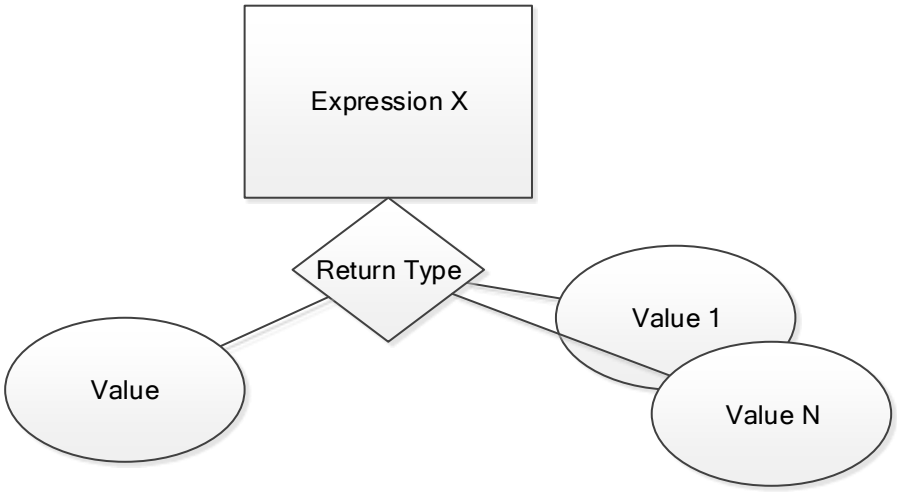
3. Let A be defined by an expression X which utilizes an expression operator EO that executes as a function on another attribute A` or references a constant value V, a list of values L, or another expression and returns a set of attributes with one specific attribute identified as a result and the other as associated attributes.



4. Let X specify a list of mapping M that links an expression operator (EO) parameter to an entity EA. Let each mapping retrieve values from entity attributes such that if M belongs to the same entity, the values correlate together for each entity instance based on the entity key and, if from different entities, they cross such that each combination is used.

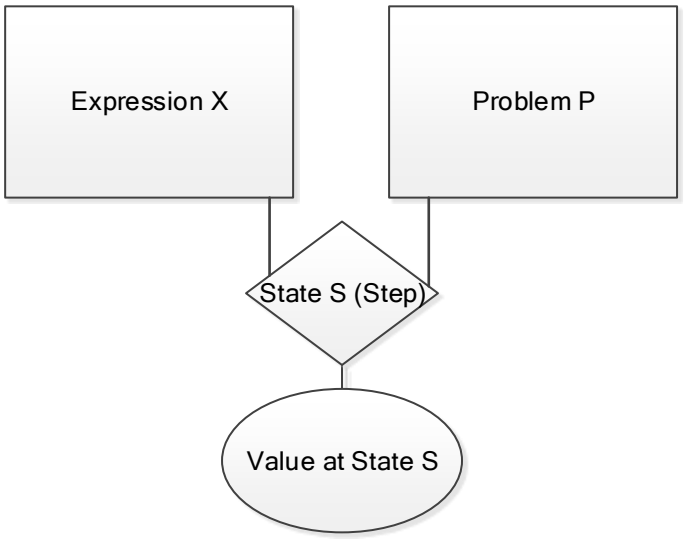


5. Let X be reflexive to EA – that is X may reference any EA and any EA may reference any X so long as the dependencies do not form a closed cycle.
6. Let X and EA return either a single value or a sequence of values of the same type. Let X return a hidden set called XH of all A associated with entity E for attribute XA representing the attributed returned by A

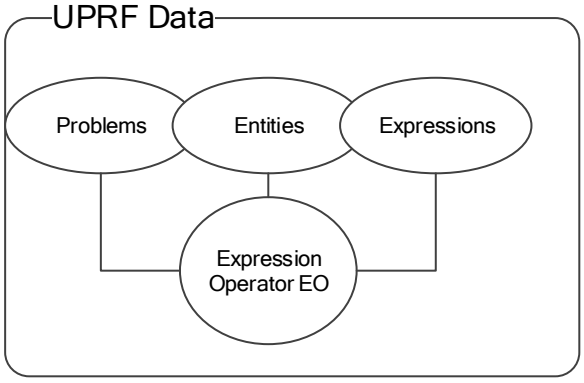


7. Let X optionally define a null expression N that references another expression returning a single value to substitute for each value instance X that is null.

8. Let X optionally utilize a state S to reference a step that defines the values those present at the state S of the problem P.



9. Let the expression operator (EO) reference any data stored within this schema as well as any data generated through operations in the system or utilize an interface to reference datasets external to the system.



10. Let Q be a query that contains expressions that can reference each other and apply compare operators upon each other as criteria C to filter results from the expression. Combine each filtered expression with other filtered expressions through a join operator JO and be known as

an extract for the query QE. Let the extract define the output attribute OA that maps to a State S.

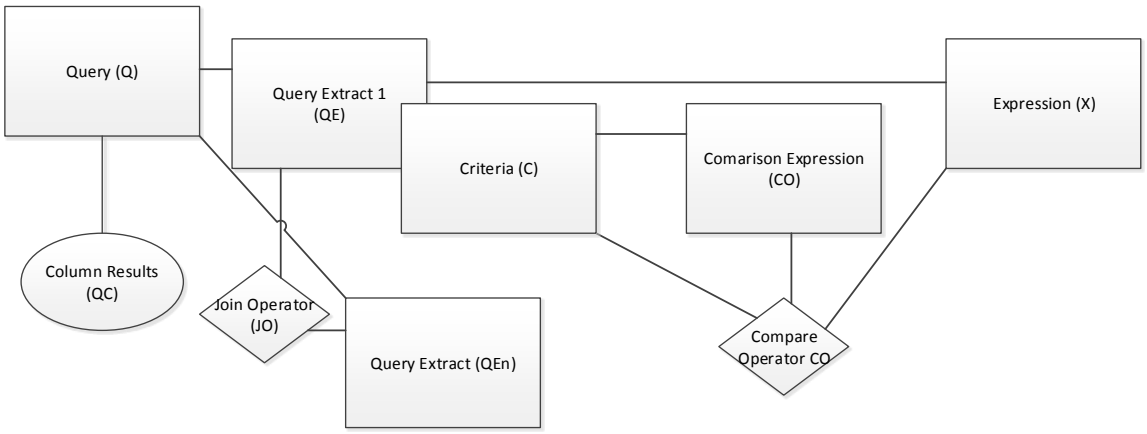
10.1. For the join operator, allow the following operations:

10.1.1. INNER retrieves values from QE passing the compare test for each attribute value from QC and drops values from a second QE where there is no match found.

10.1.2. OUTER preserves all values from both sets with null values specified for all non-matches

10.1.3. LEFT preserves all values from the expression associated to QE and only those matching to a second QE.

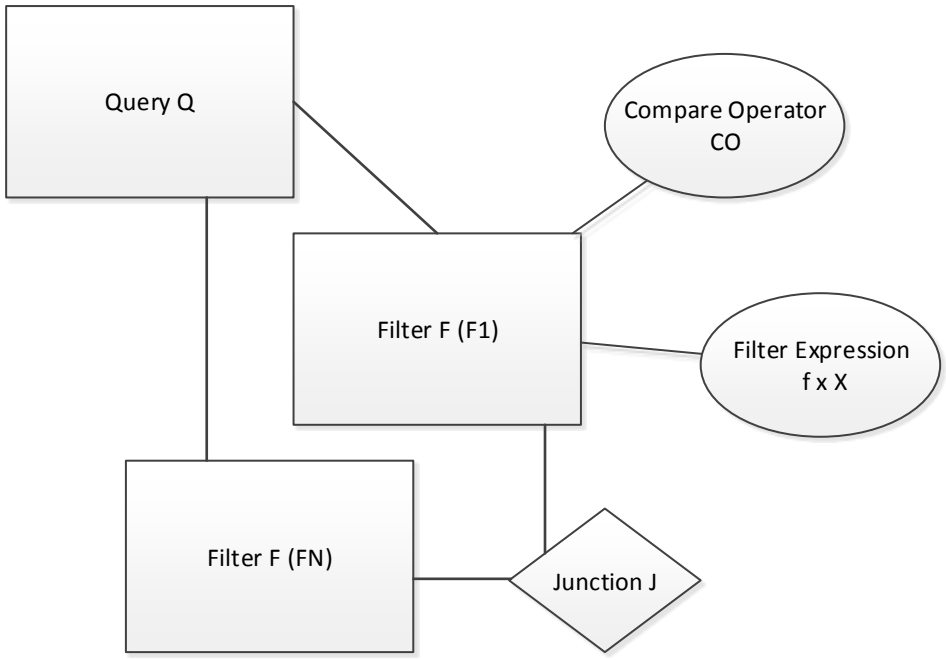
10.1.4. RIGHT preserves all values from the expression associated to the second QE and only those matching to QE



11. Let Q further contain a list of Filter F that uses a compare operator CO to compare any query column QC to any other column QC.

12. Let each filter F provide a junction to the output of any other filter in either or both an OR conjunction or an AND conjunction filter. Let this conjunction occur within the Filter F

definition such that a parent-child relationship exists to support any type of AND/OR conjunction between filters.



13. Let Problem P identify queries for projecting possible values to the following special sets that map to the required states:
    13.1. Setup Query – Query which instantiates a problem into multiple instance PI that reflect all possible starting combinations of values for the entities associated to the problem. This maps to the problem initial state PI.
    13.2. Transition Query – Query that defines all outputs that are valid for a transitive state in a problem-solving endeavor. This maps to problem transition state PT.
    13.3. Goal Query – Query that returns a status indicating achievement of a goal for an instance or overall problem and what the next action should be related to the instance or problem. A status of one indicates a solved instance that no longer requires processing; two indicates

the solution of a particular instance in scenario that calls for the continued checking of other branched problem instances. This maps to the problem goal state PG.

13.4. Fail Query – Query that returns a status indicating if a failure condition has been met for the instance (status value one). This maps to the problem Fail State PF.

13.5. Quit Query – Query that returns a status indicating the aborting of further solving efforts for any instances within the problem (status value -1). This maps to the problem quit state PQ.

Figure 4-1 depicts the overall flow in constructing a problem definition that includes queries that reference operators and attributes in order to generate the required states. The progression consists of:

1) Define a Problem P.
2) Define Attributes for problem P (A).
3) Define Entities for problem P (E).
4) Define Entity attributes (EA) based on the problem attributes linked to the entities (A).
5) Define an Expression Operator (EO) and its associated parameters (OP)
6) Define an Expression X relative to an expression operator and parameter mappings to entity attributes (EA)
7) Define a Query Extract (QE) that joins an expression to a query to contribute output attributes (OA).
8) Define a Query (Q) that integrates the Query Extracts together

9) Define a Filter consisting of Junctions (J) and Conditions (C) that filters the Query Extracts within the Query to output Query Column (QC) results that satisfy states (S) associated to the Problem P.

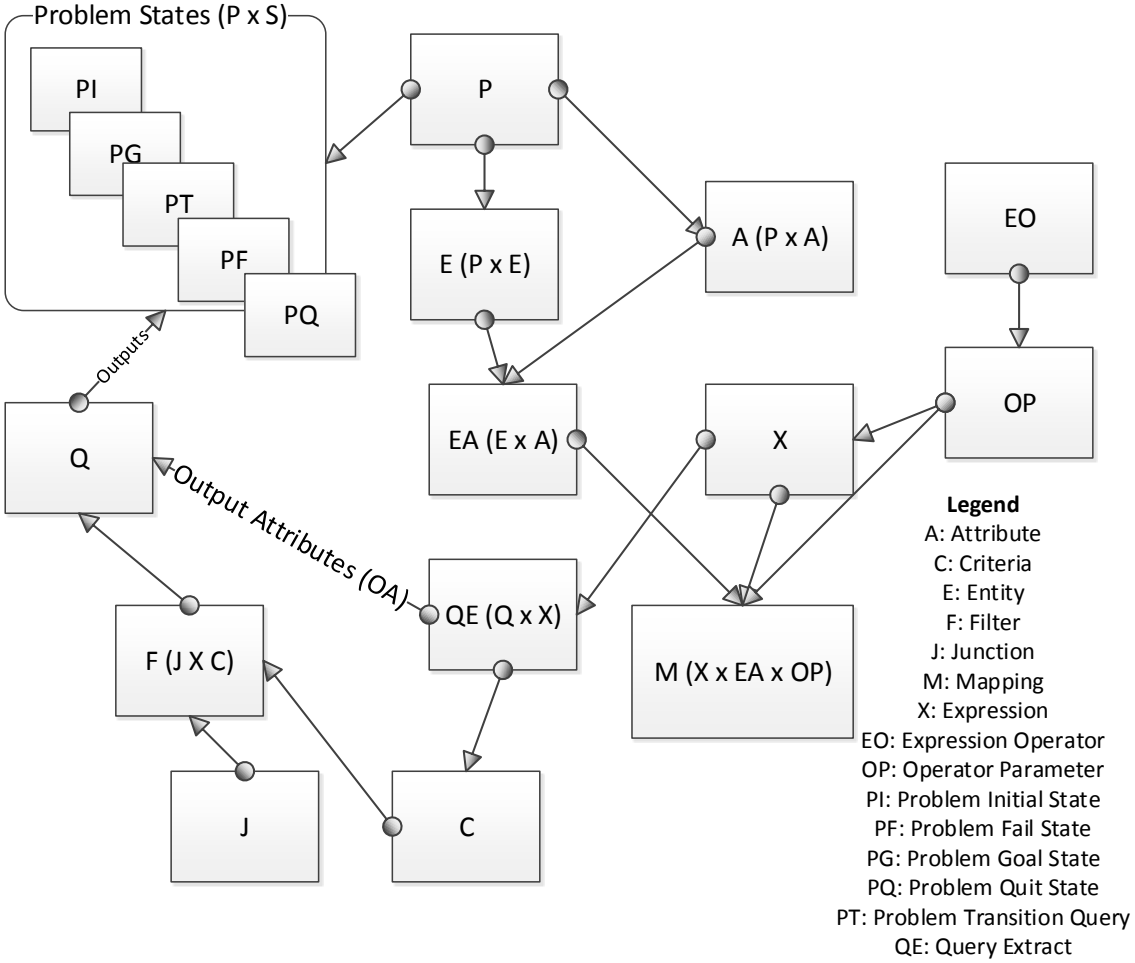


Figure 4-1: Problem Definition Flow

Figure 4-2 shows an XML problem schema that supports the problem definition. This schema originates in the database design already discussed in Chapter 3. The NullExpression refers back

to another expression, DataEntity, or attribute. DataEntity and attribute items automatically imply an expression. The expression within an extract is a reference to other expressions in a definition. JunctionGroup provides the means to associate a junction of multiple filter or extract items. Each filter or extract criteria is joined together in a Boolean fashion using the JunctionType of AND, OR, NAND, NOR, or XOR. CompareOperator supports the use of any comparison function to apply an extract result to an attribute associated with an expression. DeriveFrom provides a means to link an expression to a DataEntity or attribute to generate the range of values for the DataEntity or attribute. The Tower of Hanoi example in section 4.2 provides an illustration of how the schema applies to a problem.

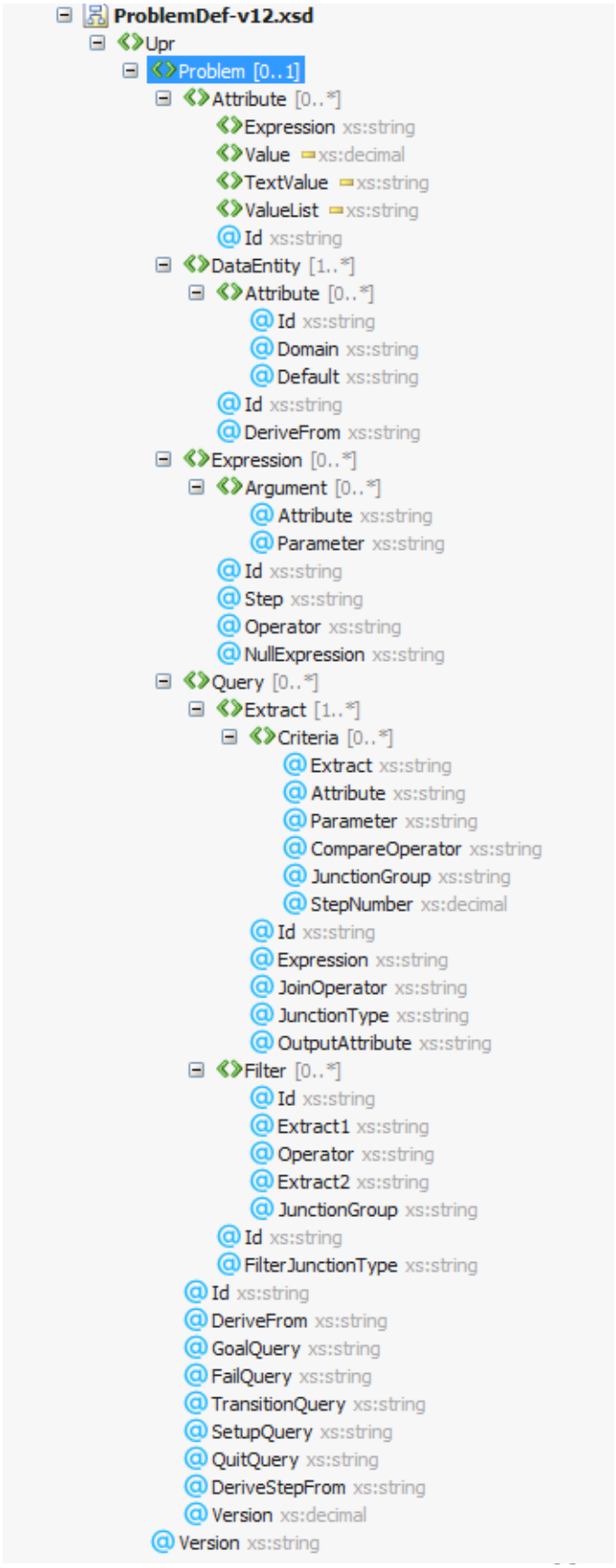

ProblemDef-v12.xsd
Upr
Problem [0..1]
Attribute [0..*]
Expression xs:string
Value xs:decimal
TextValue xs:string
ValueList xs:string
Id xs:string
DataEntity [1..*]
Attribute [0..*]
Id xs:string
Domain xs:string
Default xs:string
Id xs:string
DeriveFrom xs:string
Expression [0..*]
Argument [0..*]
Attribute xs:string
Parameter xs:string
Id xs:string
Step xs:string
Operator xs:string
NullExpression xs:string
Query [0..*]
Extract [1..*]
Criteria [0..*]
Extract xs:string
Attribute xs:string
Parameter xs:string
CompareOperator xs:string
JunctionGroup xs:string
StepNumber xs:decimal
Id xs:string
Expression xs:string
JoinOperator xs:string
JunctionType xs:string
OutputAttribute xs:string
Filter [0..*]
Id xs:string
Extract1 xs:string
Operator xs:string
Extract2 xs:string
JunctionGroup xs:string
Id xs:string
FilterJunctionType xs:string
Id xs:string
DeriveFrom xs:string
GoalQuery xs:string
FailQuery xs:string
TransitionQuery xs:string
SetupQuery xs:string
QuitQuery xs:string
DeriveStepFrom xs:string
Version xs:decimal
Version xs:string

Figure 4-2: XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<Upr xmlns:xsi="ProblemDef-v12.xsd" Version="11.0">
  - <Problem TransitionQuery="Play-Game" FailQuery="Check-if-Lost" GoalQuery="Check-If-Won"
    DeriveFrom="Disc-Count" Id="Hanoi">
      - <Attribute Id="Start-Disc">
            <Value>1</Value>
        </Attribute>
      - <Attribute Id="Min-Disc-Count">
            <Value>1</Value>
        </Attribute>
      - <Attribute Id="Max-Disc-Count">
            <Value>10</Value>
        </Attribute>
      - <Attribute Id="Start-Peg">
            <Value>1</Value>
        </Attribute>
      - <Attribute Id="Final-Peg">
            <Value>3</Value>
        </Attribute>
      - <Attribute Id="Step-Count">
            <Expression>Get-Step-Count</Expression>
        </Attribute>
      - <Attribute Id="Disc-Count">
            <Expression>Disc-Count-Range</Expression>
        </Attribute>
      - <Attribute Id="Peg-List">
            <ValueList>1,2,3</ValueList>
        </Attribute>
      - <Attribute Id="Disc-Size">
            <Expression>Disc-Size-Range</Expression>
        </Attribute>
      - <DataEntity DeriveFrom="Disc-Size" Id="Disc">
            <Attribute Id="Peg" Domain="Peg-List" Default="Start-Peg"/>
        </DataEntity>
            <!-- All attributes, problems, and entities are automatically generated as expressions and available
            for use as expressions in the database. Attributes based on constants are generated as expressions
            as needed -->
      - <Expression Id="Max-Move-Count" Step="0" Operator="POWER">
            <Argument Parameter="Exponent" Attribute="Disc-Count"/>
            <Argument Parameter="Base" Attribute="2"/>
        </Expression>
      - <Expression Id="Min-Disc-For-Peg" Step="0" Operator="MIN">
            <Argument Attribute="Disc-Size"/>
        </Expression>
      - <Expression Id="Disc-Count-On-Peg" Step="0" Operator="COUNT">
            <Argument Attribute="Disc-Size"/>
        </Expression>
      - <Expression Id="Candidate-Disc" Step="0" Operator="SELECT_NONUSED">
            <Argument Attribute="Disc"/>
        </Expression>
      - <Expression Id="Disc-Peg" Step="0" Operator="SELECT">
            <Argument Attribute="Peg"/>
        </Expression>
      - <Expression Id="Disc-Count-Range" Step="1" Operator="GENERATE_RANGE">
            <Argument Parameter="From" Attribute="2"/>
            <Argument Parameter="By" Attribute="1"/>
            <Argument Parameter="To" Attribute="Max-Disc-Count"/>
        </Expression>
```

Figure 4-3 - Hanoi Problem Definition - Part 1

Figure 4-3 represents the Tower of Hanoi problem using attributes to define the initial peg, goal peg, minimum number of discs for a simulation, maximum number of discs for a simulation. An entity (DataEntity) represents the disc identified by the size. The DeriveFrom method uniquely identifies each problem instance in terms of the number of discs for the instance. Each disc then has a peg attribute associated with it that varies from one to three (derived from the Peg-List ValueList definition). Although the Tower of Hanoi only requires a single entity and single attribute, a problem definition could have any number of entities with any number of attributes so long as each entity derives from a unique key.

```
- <Expression Id="Disc-Count-Range" Step="1" Operator="GENERATE_RANGE">
      <Argument Parameter="From" Attribute="2"/>
      <Argument Parameter="By" Attribute="1"/>
      <Argument Parameter="To" Attribute="Max-Disc-Count"/>
  </Expression>
- <Expression Id="Disc-Size-Range" Step="1" Operator="GENERATE_RANGE">
      <Argument Parameter="From" Attribute="1"/>
      <Argument Parameter="By" Attribute="1"/>
      <Argument Parameter="To" Attribute="Disc-Count"/>
  </Expression>
- <Query Id="Play-Game">
      <Extract Id="Next-Disc" OutputAttribute="Disc" Expression="Candidate-Disc"/>
    - <Extract Id="Disc-Peg" OutputAttribute="Peg" Expression="Disc-Peg">
          <Criteria Attribute="Disc" CompareOperator="EQUAL" Extract="Next-Disc"/>
              <!-- Get the disc for each peg -->
      </Extract>
    - <Extract Id="Next-Peg" OutputAttribute="Peg" Expression="Peg-List">
          <Criteria Attribute="Peg-List" CompareOperator="NOT_EQUAL" Extract="Disc-Peg"/>
              <!-- Expand to include peg list except for the current disc for next peg-->
      </Extract>
    - <Extract Id="Min-Disc-For-Disc-Peg" Expression="Min-Disc-For-Peg">
          <Criteria Attribute="Peg" CompareOperator="EQUAL" Extract="Disc-Peg"/>
              <!-- Find out what the smallest disc is for the peg of each disc-->
      </Extract>
    - <Extract Id="Min-Disc-For-Next-Peg" Expression="Min-Disc-For-Peg">
          <Criteria Attribute="Peg" CompareOperator="EQUAL" Extract="Next-Peg"/>
              <!-- Find out what the smallest disc is for the candidate pegs for each disc Use NullExpression
              to set value to disc-count for peg that has no min. This attribute effectuates an optional join
              to the expression as well as providing the placeholder value-->
      </Extract>
          <!-- We now have all information needed in the matrix to filter for next move: Example: Assume
          disc 1 on peg 2 and disc 2 and 3 on peg 1 with disc 1 moved last. Disc 1 is eliminated since it was
          moved last, Pegs for each disc eliminated in lookup that filters using NOT_EQUAL current disc peg
          Candidate-Disc Disc-Peg Next-Peg Min-Disc-For-Peg Min-Disc-For-Next-Peg 2 1 2 2 1 2 1 3 2 3* 3
          1 2 2 1 3 1 3 2 3* * Instance Disc-count applied due to use of NullExpression option -->
      <Filter Id="Filter-Disc-Too-Large-For-Peg" Operator="LESS_THAN" Extract2="Min-Disc-For-
          Peg" Extract1="Candidate-Disc"> </Filter>
          <!-- This eliminates all but row 2 above indicating the only valid next move is Disc 2 to Peg 3-->
  </Query>
- <Query Id="Check-If-Lost">
      <Extract Id="Max-Move-Count" OutputAttribute="1" Expression="Max-Move-Count"/>
          <!-- Verify that total number of moves is less than 2 raised to the number of discs -->
      <Extract Id="Step-Count" Expression="Step-Count"/>
      <Filter Id="Check-Move-Count" Operator="EQUAL" Extract2="Max-Move-Count"
          Extract1="Step-Count"/>
  </Query>
- <Query Id="Check-If-Won">
      <Extract Id="Disc-Count" OutputAttribute="1" Expression="Disc-Count"/>
          <!-- Verify that all discs are on Peg 3-->
      <Extract Id="Peg3" Expression="Final-Peg"/>
    - <Extract Id="Disc-Count-Peg3" Expression="Disc-Count-On-Peg">
          <Criteria Attribute="Disc" CompareOperator="EQUAL" Extract="Peg3"/>
      </Extract>
  </Query>
</Problem>
/Upr>
```

Figure 4-4 - Hanoi Problem Definition Part 2

```
Example: Assume disc 1 on peg 2 and disc 2 and 3 on peg 1 with disc 1 moved last.
Disc 1 is eliminated since it was moved last, Pegs for each disc eliminated in
lookup that filters using NOT_EQUAL current disc peg

Candidate-Disc  Disc-Peg  Next-Peg  Min-Disc-For-Peg  Min-Disc-For-Next-Peg
    2             1         2          2                 1
    2             1         3          2                 3*
    3             1         2          2                 1
    3             1         3          2                 3*

    * Instance Disc-count applied due to use of NullExpression option
```

Figure 4-5 - Example Matrix for Tower of Hanoi

Figure 4-4 shows the remainder of the problem definition starting from the expression that generates the size range from which the disc entity derives. The query "Play-Game" contains the extracts and filters needed to generate a result set for any particular configuration of a simulation. The query filters the data as follows:

1) Extract next disc (Next-Disc) using the Candidate-Disc expression (defined in Figure 4-4) The Candidate-Disc expression uses the function "SELECT_NONUSED" which selects any entity not selected on the prior state change. This adheres to the Hanoi rule not to move the same disc twice in a row. The selected disc is stored in the output attribute "Disc."
2) Extract next peg (Next-Peg) using the expression "Peg-List" (defined in Figure 4-3). Peg-List returns a list from one to three. This value is stored in the output attribute "Peg."
3) Identify candidate pegs for a disc by filtering out the peg it is currently on in the "Next-Peg" extract. This extract utilizes the "Peg-List" expression and filters using a comparison operator of "NOT_EQUAL" to eliminate the current peg.

4) Extract the minimum disc for the peg ("Min-Disc-For-Disc-Peg"). This utilizes the expression "Min-Disc-For-Peg" which finds the smallest disc on a peg. A comparison operator of "EQUAL" provides the filter for output from "Peg-List" results. This generates a result set identifying the minimum sized disc on each peg into a matrix. As each extract adds values to the query, intersection of prior values occurs based on the comparison operator and join operator. (The default join operator is an intersection or inner join, but supports all standard join types – (see # 10 under the Postulates).) Figure 4-5 shows a sample result matrix.
5) Extract the minimum disc for the peg a second time, but this time utilize the "Next-Peg" as the criteria for filtering. This determines the minimum-sized disc on only the candidate pegs (not the current peg of the disc) which ultimately determines if it possible for the disc to be moved to the peg by the filter. Note that the Min-Disc-For-Peg includes a NullExpression construct that forces the value to the max-sized disc (Disc-Count) if no discs are on the peg.
6) As shown in the matrix in Figure 4-5, this query provides a matrix with rows and columns, which allows further filtering.
7) The expression extracts have at this point reduced the results to eliminate the following:
    a. Exclude the peg that the disc is currently on in the Next-Peg column.
    b. Exclude any disc that moved on the last move.
8) The result set still contains invalid moves without an additional filter to ensure that no larger disc moves on top of a smaller disc. The filter "Filter-Disc-Too-Large-For-Peg" eliminates any disc that is not less than the smallest disc on the candidate next peg with

the empty peg condition. This ensures that the result for an empty peg returns a disc number higher than the highest due to the NullExpression construct thus permitting the peg to receive any disc.

9) Given the above, the Play-Query provides only valid next moves for the simulator.

10) The final step is to define the queries that define if the simulation is in a lost or won state so that the simulation instance does not go on forever and to provide a goal for the simulator to pursue in the solution. The lost query ("Check-If-Lost") verifies that the maximum number of state changes (moves) has been exceeded which is defined by the Max-Move-Count expression as the number two raised to the number of discs. For example in a three-disc scenario, the solution is achievable within seven moves, for a four-disc scenario, fifteen moves, etc. The goal step defined as "Check-If-Won". Check-If-Won utilizes the expression 'Disc-Count-On-Peg" to compare if the number of discs on a peg is equal to the Disc-Count associated to the problem for the peg equal to the value "Final-Peg" which has a value of three. In other words, the simulation is successful if the number of discs on peg three is equal to the total number of discs. The return value of "1" indicates to the simulator that the simulation can be marked as successful for the problem instance. At this point, there is no need for further simulation activity for the problem instance. Additional return values provide support for other problems that may have multiple solutions by allowing "2" as a return attribute. Return attribute "2" directs the simulator to mark a simulation instance as complete but to continue searching for additional solutions. For the Tower of Hanoi, there is only one optimal solution path for any

configuration of discs, so the simulation stops for each instance as soon as it finds the solution.

4.1.1.2 Claims from Postulates

1. Based on the dynamicity of the expression operators and the ability of the expressions to operate against any set of attributes, defining the appropriate functions can derive any set of values associated with a problem.
2. Any problem state where state is the result of a query that relate behaviors of objects within a problem is feasible based on the unlimited join structure that supports all set operations and filtering constructs combined with the ability for any expression operators to be loaded into the framework. This derivation is context sensitive, related to other states using the state sequence qualifier for expressions or when referencing expressions.
3. This definition supports the ability to generically define any problem using the same schema and utilize a generic simulation approach to apply the functions in order to generate states in the pursuit of a goal query defined for the problem.

4.1.2 Reflexive Property of Problem Solving

The reflexive property of problem solving allows the solution to a problem to be query-able within the constructs of the same schema that defined the problem. This provides the foundation for transforming a problem that has associated solution instances into a higher order problem that attempts to augment the base problem with a prediction (transform) operator. The prediction operator then generates additional expressions and queries to reduce the number of simulations and predict the solution path for additional instances of the problem. The property conforms to the following two constructs:

1. Any problem P that the framework presents for solution by the system generates multiple outputs for entities along a state sequence. The capture of states representing all values of all entities throughout solution endeavors can be stored generically through the same entity/attribute structure as that used to represent the problem. This means that the entire solution endeavor is visible to a higher order problem using the same schema.
2. All problems are solved using a generic process such that the output actions taken in regard to each problem are stored generically within the same entity/attribute structure which is represented by:
    a. Problem Instance
        i. Problem-Identifier
        ii. Instance-Identifier
        iii. Result-State
        iv. Instance-Step
    b. Entity Instance
        i. Instance-Identifier
        ii. Instance-Step
        iii. Entity-Key
        iv. Entity-Attribute
        v. Value

Based on these constructs, UPRF captures all of the states of each entity instance associated with a problem for each entity attribute. The next step is to define a method for querying the

problem information and the solution paths such that instances solved via simulation provide the basis to predict values for additional instantiations of the same or related problems.

4.1.3 Postulates of the Predictive Solving Framework

14. Let VS be a set of values over a series of steps that represent the truth or falsity for an activation of a particular value for the entity at a given state. The simulation system that operates against the schema from 1.2.1 for every entity instance value generates the VS utilized in a problem.
15. Let PTO be a prediction transform operator defined as a function that receives parameters from a problem P, an entity-instance attribute-value sequence (PVS) to predict. The PVS is the last value of the prediction for this operator for the target VS (TVS) and a source entity-instance-attribute-value sequence (SVS) to use as a source. Let the output of the PTO be a value sequence and let each instantiation operate upon the prior value of the PVS
16. Let PTO reference only the information passed to it by the problem definition, that is, source instances containing entity-attribute-value sequences for earlier instances of the problem. Require that PTO record all sources and targets utilized to derive a VS.
17. Let the application of each PTO result in a new step to be accomplished in the same way as any problem simulation and the combination of values based on the selection of values be captured as another value sequence

4.1.4 Claims based on the Postulates including 1.2.1 and 1.2.2

1. The higher order problem is representable in the same problem schema that represented the lower level problem without loss of any fidelity since it uses the same entities as those of the lower order problem. The prediction operator belongs to a type of expression operator, but the particular operator selection materializes as an attribute value in a standard entity-attribute structure. This

allows for the correlation of one operator sequence from one solving instance to operator sequences from another solving instance enabling creation of a transformation problem for detecting a higher-order sequence for transforming the lower level instances.

2. The higher order solution requires the same solving approach as the lower level instance meaning the process is recursive and allows continually higher-order use of the same techniques that the system found successful in pursuit of a lower order problem.

4.1.5 Transformative Property of a Problem Solution

Based on 4.1.4, there is a transformation available to generate a higher order problem from any lower order problem. This includes generating higher order problems from the second-level higher order problems without limit. This means that the problem solver is able to use simulation to optimize the solution discovery phase for not only a base problem but also for the process of optimizing solution discovery. This recursive model can scale up infinitely. The transformative property concerns the transposition for utilizing the known solution states of a problem and transforming this to a higher order problem with the goal of testing the prediction operator's success at predicting the outcome based on prior instances. If the transformation process works for a lower order problem using the same simulation model for solution discovery then it must also be transformative to a higher order problem. This section provides proof that the second-order transformative problem is identical to the third-order problem and all higher order problems. The proof accomplishes this outcome by pivoting of the problem upon the steps involved for finding the optimal application of prediction operators against combinations of input sequences over a sequence of application. This application generates sequences for each higher and higher order

problem that follow the same simulation model and use the exact same schema definition. From this:

- The information about the solution path for each instance thus presents as a higher order problem with the goal of learning from the solution pattern for prior instances in order to predict the solution pattern to apply to additional instances. The higher order problem for any lower order problem incorporates the entity/attribute structure representing information about the problem instance and adds the following for each entity instance (defined by instance-identifier, instance-step, entity-key, and entity-attribute):
    - Entity-Instance-Prediction
        - Predicted Instance-Entity-Attribute
        - Predicted Value
        - Instance-Step
        - Prediction Operator from the domain of possible predictors
    - Entity-Instance-Prediction-Sources
        - Source Instance-Entity-Identifier-Attribute from the domain of all instances earlier than the targeted instance.
        - Source-Step from the domain of steps
    - Predictor-Operator-Query-Addition - Incorporates additional filtering to reduce the size of possible solutions
    - Predictor-Operator-Expression-Additions – Additional expressions to support the updated query.

- The above then provides for a higher order simulation that generates branches for each prediction-operator and the combinations of source instances associated with source steps. The predictor operator itself is an entity-attribute and generates a value sequence. The goal of the higher order problem then becomes selecting from the same set of prediction operators from the lower order problem that accurately predict the value sequences that reflect the optimal sequencing of the prediction operators against the source instances.
- The solution goal for the higher order problem is to identify the prediction-operator that most effectively predicted the values. This is modeled as a goal of the higher order problem in terms of the following query constructs:
    - Find the sequence of prediction operators that generated the query additions that filtered the solution path and resulted in the fewest steps to achieve the goal state of the underlying problem.
    - Map this sequence of prediction operators as the higher level goal for the higher order problem such that prediction of lower instances derive by the solution sequence of the higher order problem. Once again, the selected sequence of prediction operators that resulted in the selection of the lower order prediction operators can generate the correct solution path.

4.2 Proof by Example – the Tower of Hanoi

This section examines the solution sequences of various instances with different numbers of discs for the Tower of Hanoi and the transformation process for pivoting a solution sequence to become a higher order problem. The transformative process itself models itself as higher and

higher order sequence problem. Ultimately, the framework converges to a generic solution sequence.

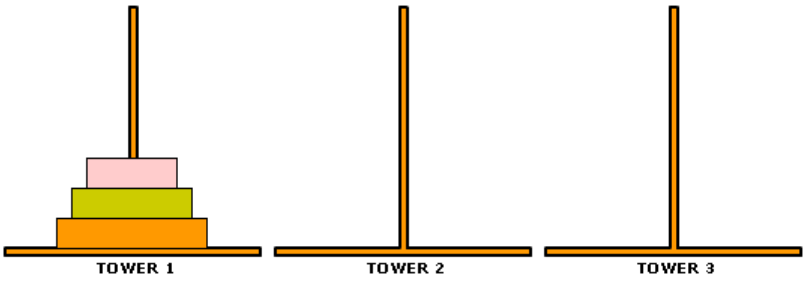


Figure 4-6: Three Disc Tower of Hanoi

4.2.1 Methodology

1. Define the Tower of Hanoi problem as illustrated by Figure 4-6 using the generic problem-solving schema (Figure 4-2).
2. Generate instantiations based on the setup query for the range of discs defined for simulation.
3. Utilize simulation to process the outputs of the transition query until achieving success for four instances of Hanoi (two to five discs). Table 4-1 illustrates results of the entity-attribute value sequences.
4. Using the higher order problem (Figure 4-10), apply prediction operators that evaluate the solution sequences, generate additions to the lower level problem queries, execute the queries, and measure the success. This process runs as a simulation and results in identifying the sequence of prediction operators necessary to solve each larger instance from the smaller instances.

5. The entity-attribute value sequences for the optimal solution evolve from instances three and four. The higher order problem for instances three and four finds the causative predictive operator that involves a transformation based on even or odd discs. Instance five applies this transformation to predict the sequencing of instance six.

4.2.2 Proof from the Data

Table 4-1: Sequence Patterns from Hanoi by Disc and Peg

Discs	Disc	Peg	Move States
2	1	1	---
2	1	2	1--
2	1	3	--1
2	2	1	---
2	2	2	---
2	2	3	-1-
3	1	1	----1--
3	1	2	--1----
3	1	3	1-----1
3	2	1	-------
3	2	2	-1-----
3	2	3	-----1-
3	3	1	-------
3	3	2	-------
3	3	3	---1---
4	1	1	----1-----1----
4	1	2	1-----1-----1--
4	1	3	--1-----1-----1
4	2	1	---------1-----
4	2	2	-----1---------
4	2	3	-1----------1-
4	3	1	---------------
4	3	2	---1-----------
4	3	3	----------1---
4	4	1	---------------
4	4	2	---------------
4	4	3	-------1-------
5	1	1	----1-----1-----1-----1-----1--
5	1	2	--1-----1-----1-----1-----1----
5	1	3	1-----1-----1-----1-----1-----1
5	2	1	---------1----------1---------
5	2	2	-1----------1----------1-----
5	2	3	-----1----------------------1-
5	3	1	------------------1----------
5	3	2	----------1------------------

5	3	3	---1----------1---
5	4	1	----------
5	4	2	-------1----------
5	4	3	----------1-------
5	5	1	----------
5	5	2	----------
5	5	3	----------1----------

In the above chart, each disc has a relationship to the peg affected on a particular move. A sophisticated learning algorithm should be able to find the patterns between the discs and pegs, but a simpler method emerges using aggregation. Rather than analyzing the detailed patterns for each combination, the result of simply aggregating all the states across the sequence for each distinct value for the entity (disc) and for the attribute (peg) without regard to the intersection of the entity and attribute provides more concise insights as shown below:

Table 4-2: State Sequence by Disc

Discs	Disc	State Change Sequence
2	1	1-1
2	2	-1-
3	1	1-1-1-1
3	2	-1---1-
3	3	---1---
4	1	1-1-1-1-1-1-1-1
4	2	-1---1---1---1-
4	3	---1-------1---
4	4	-------1-------
5	1	1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
5	2	-1---1---1---1---1---1---1---1-
5	3	---1-------1-------1-------1---
5	4	-------1---------------1-------
5	5	----------------1----------------

Table 4-3: State Sequence by Peg

Discs	Peg	State Change Sequence
2	1	---
2	2	1--
2	3	-11

3	1	----1--
3	2	-11----
3	3	1--1-11
4	1	----1----11----
4	2	1--1-11-----1--
4	3	-11----11--1-11
5	1	----1----11-----1--1-11-----1--
5	2	-11----11--1-11-----1----11----
5	3	1--1-11-----1--1-11----11--1-11

A simple sequence recognition algorithm can spot that that the pattern for the smallest disc is to change state every other move, for the next largest disc to change every fourth move, the third every eighth interval and so on. When the peg states are also examined for patterns in disc counts three to six, only a few of the patterns need to be identified and the rest can be deduced due to the standard rules of play. This effectively generates a rule that only allows one choice so that the sequence can generate instantly. From there, all that is necessary is a reversal process to generate the query criteria that reflects the relational sequences.

To prove that the transformative property works, the framework must show the following three things:

1. The learning problem process for finding a sequence of machine learning algorithm operations can be defined schematically using the same schema and operations associated with the base problem;
2. There is a problem definition P’ which represents the problem of searching for patterns to find the optimal solution;
3. The process for the generating and solving the learning problem P’ spawns a P’’ to optimize the learning problem P’ and that P’’’ is generated in the same way as P’. This indicates that there is no limit to the learning depth obtainable from higher order problems.

An additional consideration is that if the system truly generates higher and higher order problems, there must exist an energy function to govern the levels of problem solving at which the diminishing returns of the computation outweigh the benefits of reflection. A human reflective analogy of this is that one may think about how well they performed a particular task to try to identify improvement and about how to think about the mechanism for evaluating such performance – a process that can go on indefinitely. At some point the human mind implicitly imposes a restriction on reflection in order to permit functionality and not spend all of one's time simply reflecting upon reflecting. Therefore an additional constraint to govern system behavior is:

4. There is an equilibrium process for the efficiency problem to determine the number of levels is definable in terms of an overriding problem in the system governed by a least-energy function EF.

Figure 4-10 illustrates a schema for recursive self-examination of problem solving to support the continuous-learning goal. The key to the functionality lies in the expression operators' extensibility to select data from within the schema without requiring explicit redefinition of the optimization problem in terms of a new entity. Referencing existing data generated from the simulation solution paths provides the optimization problem with the information necessary to execute operations to try to meet the goal.

However, allowing the optimization problem to self-reference data within the system schema that targets optimal discovery of the solution creates a problem if such an optimization problem does not adhere to the constraints of the framework as far as generating solution data for the higher order problem. To resolve this, the framework uses the same simulation process to find the optimal solution path for a problem as that for probing a solution to a base problem (founded on the given

constraints). This means that the effort of solving the optimization problem generates data about the optimization problem such that the solution steps for the lower level optimization problem now become the inputs for the higher order solution problems. Once the frameworks shows that the same process used to generate the first-order higher-level solving problem is the same as the process used for generating the higher-order problems, there is proof that the learning process is recursive. The process of finding each higher order problem generates optimizations back to the lower level problem such that new instances of the lower level problem are solved using the optimizations generated from the higher order problem. In the case of Hanoi, this manifests itself through the generation of the learning problem map that enables the new instances of the problem to benefit from the optimization query generated by the higher order problem.

4.2.3 Summary

Based on this, the following steps emerge for problem resolution:

1) Identify a problem P that has multiple instances each increasing in size such as the Tower of Hanoi with more and more discs added.

2) Represent the overall problem using relational algebra to define unique entities with their attributes that define the problem.

3) Define expressional functions to return possible domains of values including aggregates and queries that follow a relational model that joins the expressions and filters the results with Boolean and/or conjunctions to represent the valid states for entities/attributes which create an instance, define the valid transitions, and attain a goal state.

4)Solve the instances of increasing size in order using a generic simulator. Generate a relational state sequence corresponding to each attribute within the problem instance and all possible values and the sequence at the activated value.

5)Within each simulation instance, generate instances of a prediction problem based on the same schema as that for the base problem. Reference as entities the state sequences from within the instances and transform operators for application of each state sequence to predict future values of the state sequence. For example, given Hanoi simulation instance S1 with two discs, select operators for each branch of the instance in a prediction problem that predicts the next value in the state sequence for each state sequence change within the instance. Allow the prediction operators (also known as transform operators) to be visible not only to the sequences associated with the current instance but also to the sequences from prior solved instances. This means that while framework solves simulation instance S2 with three discs, the prediction operators can reference solved sequences from S1 and apply the prediction operators against the earlier sequence to generate the expected sequence values for S2. Allow each subsequent instance to reference the solved sequences of the prediction problem associated with each instance. Further, allow prediction operators that may generate portions or the entire section of the state sequences rather than just one value at a time. Set the goal for each of these prediction instances to generate the solution instance in the fewest number of steps.

7)Once there are two solved instances of the first-order prediction problem, generate a prediction problem that references the lower-level prediction problem-state sequences as entities in the same way as process five above. Continue to generate more high-level prediction problems while creating the lower-level generation instances. Expand the scope of the higher level prediction

to include instances from other problems since at this point the focus is on improving the higher order solution selection process.

8)Continue processes five through seven with higher and higher transformations until reaching a point of diminishing returns in terms of effort and success. This problem is an ongoing problem that checks the number of operations against the success ratio and dynamically injects constraints that prevent problem escalation to the point of diminishing returns. This is the equilibrium problem.

Figure 4-6 illustrates increasing problem transformation depth as the framework solves more simulation instances for the Tower of Hanoi. The first-order problem is to determine the transformation operators that vary the sequences of a source instance so that they match a prediction instance (target). Once there are two solved transformation solutions, a sequencing problem arises that determines the sequencing operators to act upon one transformation instance to create the transformation operations in a prediction instance. Once there are two solved sequencing solutions, a sequence generation problem learns the operators needed to generate the sequencing operations in a predicted instance from a source instance.

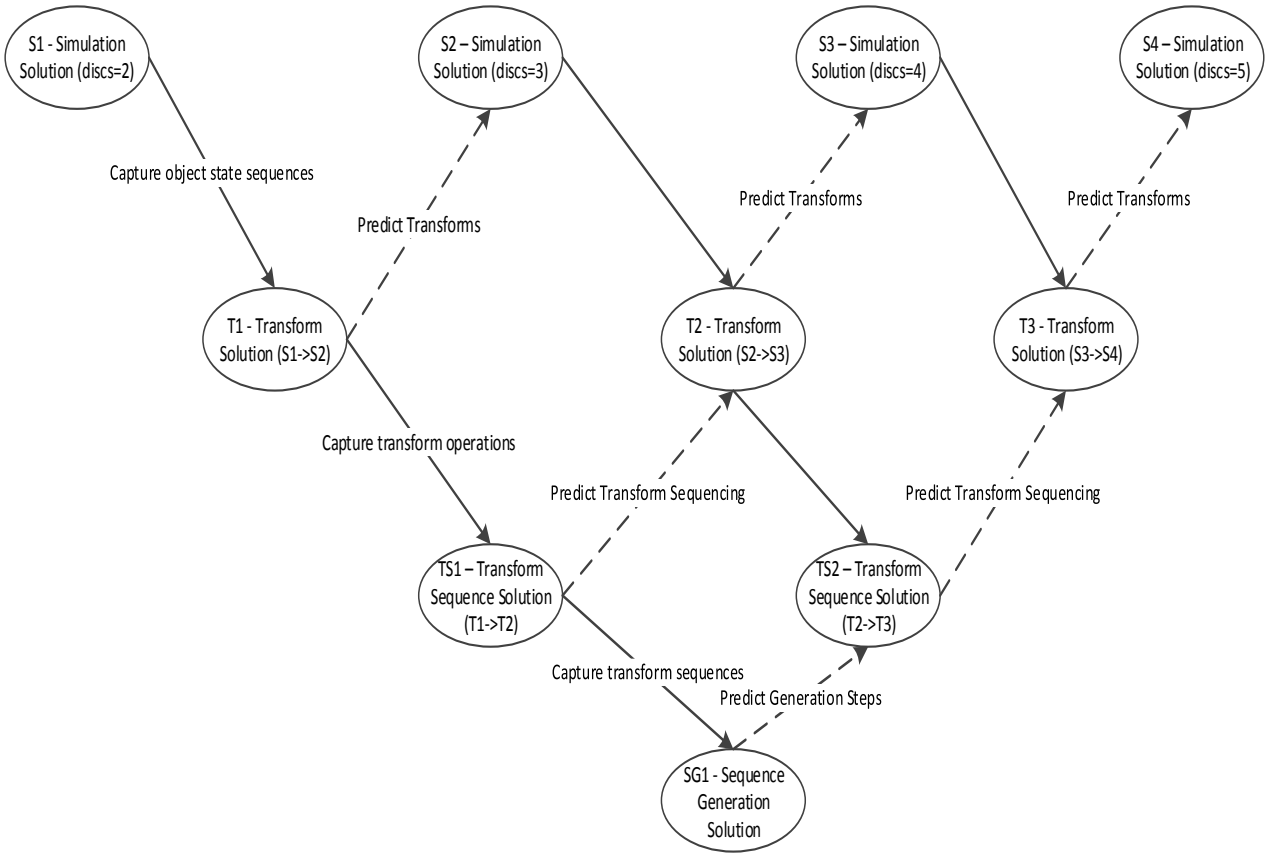


Figure 4-7: Simulation Graph

At this point in the Tower of Hanoi, a repeating sequence becomes evident and the sequence generation operators are able to generate the solution to a higher instance. The framework accomplishes this using the lower solved instance without requiring the use of simulation to increase the breadth or depth of the established solution space. Rather than relying on simulation, the tree can be co-recursively visited, creating a new transformation sequence problem instance to leverage the last transformation solution as input to generate a transformation instance, which then generates the simulation output result without re-simulating. This process repeats using the output simulation of each instance with higher disc-counts until solving the desired disc-count. Figure 4-7 illustrates the solution generation process wherein the sequence generation solution is able to

generate the next transform sequence solution. This in turns generates the next transform solution, which then generates the output that would have come from simulation. To verify that the generated solution is correct, the framework can execute further simulations. Each simulation adds further depth of learning such that the generational sequence solution evolves into higher-level generational sequence solutions. As the learning depth increases, it becomes more and more likely that a general case solution will evolve for a deterministic scenario. For the Tower of Hanoi, four simulations that spawn three levels of solution generation are adequate for the transform operators to identify a pattern that works for all instances of disc counts.

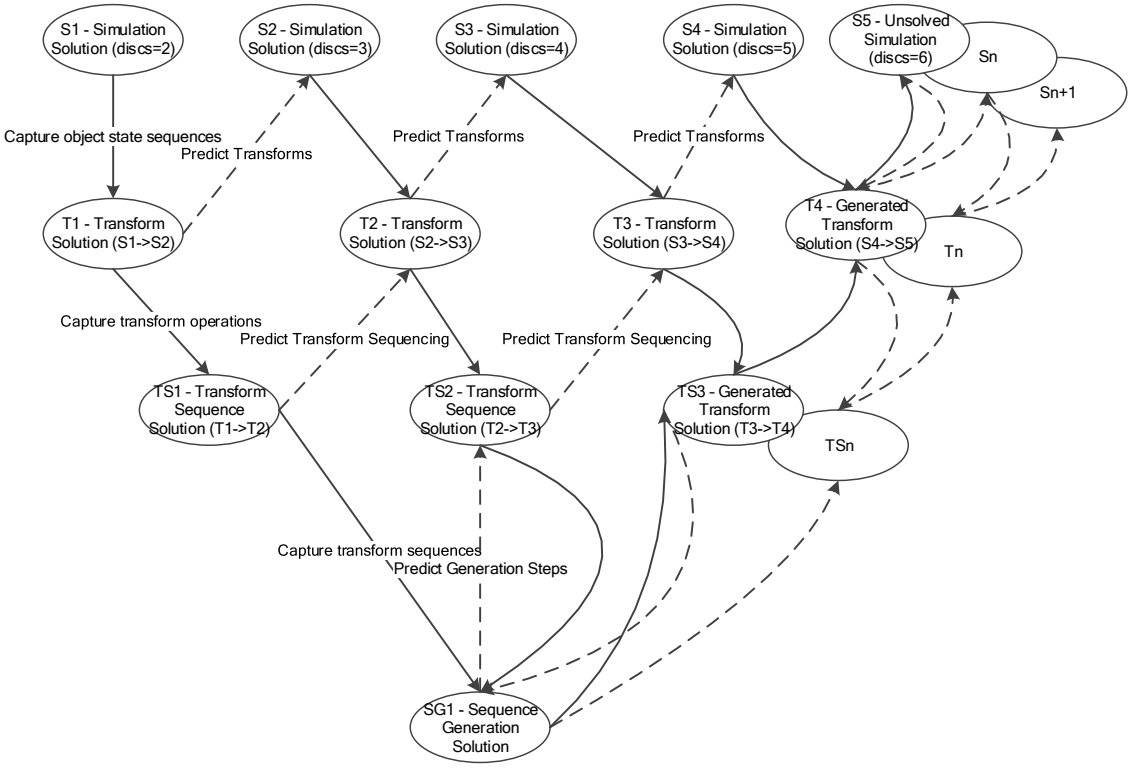


Figure 4-8: Reversibility Operation to Generate Solution Instances

Table 4-4 illustrates the generation of sequences associated with activation states from solved simulation instances. The framework exposes these sequences for solving using transformation operators to calculate the sequence of one instance from a lower instance. The framework actually identifies the transformation needed for the discs and pegs on the very first transformation solution using the first two instances. The complexity here comes from the fact that each instance introduces an additional instance of the disc entity based on the disc-count for the instance. In order to generate a transformational sequence solution that incorporates this additional entity, sequence results must be utilized that are then aggregated and correlated so that the copying of the prior entities as well as the creation of the entities are flattened as distinct sequences.

Table 4-4: Solution State Sequences for Disc-Count = 2 and Disc-Count=3

Attrib.	Value/Expression	Sequence/ Transform Results
S1 - Solution path from simulation for Disc-Count=2		
Disc	--1 (1)	1-1
Disc	Min-Disc	1-1
Disc	Min-Disc+1	-1-
Disc	-1- (2)	-1-
Disc	Disc-Count	-1-
Peg	--1 (1)	---
Peg	-1- (2)	1--
Peg	-11 (3)	-11
S2 - Solution path from simulation for Disc-Count=3		
Disc	--1 (1)	1-1-1-1
Disc	Min-Disc	1-1-1-1
Disc	Min-Disc+1	-1---1-
Disc	-1- (2)	-1---1-
Disc	Min-Disc+2	---1---
Disc	-11 (3)	---1---
Disc	Disc-Count	---1---
Peg	--1 (1)	----1--
Peg	-1- (2)	-11----
Peg	-11 (3)	1--1-11

Table 4-4 illustrates the relational state sequences from two instances of a Hanoi solution capturing the entities (discs) and attributes correlated to the activation of particular values in a solution sequence. For example, in the two-disc simulation, the disc with the smallest size (one) activates on the first and third move of the solution while activating the larger disc on the second move. Likewise, each peg correlated to a value is activated at different points, with peg two activated on the first move (disc one moves to peg two) and then peg three activated for the remaining two moves. Within UPRF, relational state sequences should be fully reversible; the combination of all captured states for any instance of a problem can be reversed to replay all of the actions involved in a simulation.

The second simulation on Table 4-4 shows the relationship between the state sequences associated to a second instance with an additional disc. The color-coding illustrates how the sequences from the first simulation constitute sequences in the second simulation solution. For example, the smallest disc inherits the pattern of moving every other move; the next largest disc moves every fourth move; and the new disc takes on the same pattern as the largest disc in the first simulation within the extended sequence.

The peg sequences also show a relationship that maps sequences from multiple pegs to generate new relationships. For example, the peg one sequence of the second instance derives from the sequence for peg one followed by a zero bit and appended with the sequence for peg two from the first instance. This pattern continues for the other pegs serially with each peg instance and the second peg of instance two derivable from the instance-one peg-three sequence plus a zero bit and then rotating back to append the peg one sequence. The framework can create these transformations using a bit function as shown in Table 4-5.

In Table 4-5, the attribute sequenced is a transform operator. The transform operator represents an algorithm for generating or updating a target sequence from a source sequence. For example, the sequence for the smallest disc from the two-disc simulation initializes with a Copy-Segment operation to copy the sequence to the smallest disc in the three-disc instantiation. The framework applies the Copy-Segment operation twice to the target disc with an intervening zero bit insertion. These operations together generate the required relational state sequence for the first disc. The process repeats for each disc with the new disc utilizing a different set of operations taken from the largest disc in the first instance.

The transform operators themselves become relational state sequences that identify which operators the framework utilizes across the entire solution sequences rather than only for the source/target instances. De-coupling the source object activations used by the sequence into separate sequences provides traceability of these sequences to enable reversibility. Relational sequences of type "sequence" represent both the operator sequences and the entity/attribute activations associated to the operator sequences.

Note that the purpose of the framework is not to inherently provide the logical functions needed to achieve the transformations but to provide an environment for hosting machine-learning algorithms that serve as transform operators. The framework provides the same simulation problem solving approach for finding the learning operations as that for playing out a simulation through brute force. The goal is to find some function that accomplishes the pattern recognition using one or more source instances. These source instances predict the correct relational state sequence in an unsolved instance. The framework also ensures that each algorithm executes with traceability back to the parameters used by the algorithm to make the prediction. This ensures that

the execution of all algorithms is constrained within a measurement framework so that the operation of the algorithms themselves make themselves visible as higher-level optimization problems. This also provides the potential to analyze algorithms for correlation to rank the selection processes which the process thereof continually improves as the system solves more problems. The algorithms themselves are stored as data entities in the framework to allow extensibility. The framework can register any algorithm for use as a transform operator within the framework utilizing the prescribed interface for the algorithm to search the source values exposed for a problem instance and record the sequences of operations utilized to achieve a target sequence.

Table 4-5: Transformation for New Instance Using a Prior Instance (Count=3)

Attrib.	Value/Expression	Sequence/ Transform Results
T1 - Solution path for pattern for disc-count=2 to solution for disc-count=3 (S1->S2)		
E-Operation	Copy-Segment	1-11-1-
E-Operation	Insert-Bit	-1--1--
New-E-Operation	Binary-Expand	------1
E-Source	Instance-1:Disc-1	1-1----
E-Source	Instance-1:Disc-2	---1-11
E-Source	Instance-1:Max-Steps (Power(2, Disc-Count) -1	------1
E-Target	Instance-2:Disc-1	111----
E-Target	Instance-2:Disc-2	---111-
E-Target	Instance-2:Disc-3	------1
E-Source	0	-1--1--
Attribute Operations		
A-Operation	Copy-Segment	1-11-11-1
A-Operation	Insert-Bit	-1--1--1-
A-Source	Instance-1:Peg-1	1--1-----
A-Source	Instance-1:Peg-2	--1--1---
A-Source	Instance-1:Peg-3	------1-1
A-Source	0	-1--1----
A-Source	1	-------1-
A-Target	Instance-1:Peg-1	111------
A-Target	Instance-2:Peg-2	---111---
A-Target	Instance-3:Peg-3	------111

Table 4-5 shows the three different types of operations and how they correlate to source/target entities and attributes. The three operations are:

1. E-Operation: Carries out a transformation that uses source entities and affects a target entity.
2. New-E-Operation: Carries out a transformation that uses sources entities and creates a new target entity.
3. A-Operation: Carries out a transformation that relates to source and target attribute values.

In the above example, the Copy-Segment operation utilizes Instance 1/Disc 1 on steps one and three while the Copy-Segment affects Instance 1/Disc 1. Intervening between the uses of the source Instance 1/Disc 1 is the use of the Insert-Bit function which is active at step two to insert an intermediate bit in the target Instance 2/Disc 1 sequence. The peg sequence population follows the pattern required to generate the peg two sequence for the second instance from the first instance by first activating the peg one sequence, then inserting a zero bit, and finally adding the sequence from peg two, also from the first instance. This repeats for the second peg on the second instance. However, the third peg involves the insertion of a one bit between the two sequences. There is also an exception to the source entity handle outlined in the New-E-Operation. This operation utilizes a binary expansion to generate the sequence for the new disc both appending and prepending the source sequence by enough bits to fill out the required sequence length. The required sequence length is understood in terms of the expression that calculates the maximum move count. For example, applying the move count as the number of bits expands the sequence for the largest disc two in a two-disc scenario from 010 to the large disc three sequence for a three-disc scenario of

000100 since the number of moves in a two-disc scenario is three and for a three-disc scenario it is seven (Power (2, Disc-Count) – 1). In both cases, the number of total bits to expand symmetrically with zero fill is the number of maximum moves.

The transformation sequence applies to additional instances (as shown in Table 4-6) wherein the simulation pattern from an instance of three discs can predict the pattern for a four-disc solution. Note that the transformation operations are the same except for the addition of further operations to support the creation of the new disc associated with the instance for the disc-count of four. It is clear with the third simulation that a solution pattern exists which involves simply using the prior transformations and adding in the new transform for the new disc. This creates a temptation to stop the process and simply implement a programmatic solution. However, the goal of UPRF is not only to identify the solution pattern, but also to generate a pattern that implements the solution transformation. Therefore, the framework continues onto the next level of identifying the transformations for creating the transformation sequences for the original problem.

To complete the next step, a new simulation for S5 is necessary in order to spawn another Transformation instance T3 so that a second instance of the transform solution (TS2) wherein the pivot that includes the operation type is incorporated. Having TS1 and TS2 transform solution paths enables the highest order of problem solving which handles transform operation sequences generically shown as SG. The transformation sequence for TS1 and TS2 incorporates the base expressions that reflect data about the instance itself, specifically the disc count in order to calculate the number of offset operations required to achieve the transform from TSN to TS-N+1. The generic solution is thus de-coupled from the specific instances and able to operate on any simulation instances in an identical fashion. This allows execution of the higher order transform

even where no supporting simulation exists. The capability represents the pivot point in the recursion, such that co-recursion reverses back down the tree and ultimately generates the next simulation solution instance without the need to carry out simulation, but only to implement the transformations. Thus, instead of requiring exponential complexity to explore the solution space, the complexity is linear with respect to finding the solution approach based on the number of discs. This does not mean that the problem itself reduces to linear complexity, as the size complexity still must increase with larger simulations to reflect the need for an exponential increase in moves for each added disc.

Table 4-6: Solution State Sequences with Disc-Count = 3 and Disc-Count=4

S2 - Solution path from simulation for Disc-Count=3		
Attrib.	Value/Expression	Sequence/ Transform Results
Disc	--1 (1)	1-1-1-1
Disc	Min-Disc	1-1-1-1
Disc	Min-Disc+1	-1---1-
Disc	-1- (2)	-1---1-
Disc	Min-Disc+2	---1---
Disc	-11 (3)	---1---
Disc	Disc-Count	---1---
Peg	--1 (1)	----1--
Peg	-1- (2)	-11----
Peg	-11 (3)	1--1-11
S3 -Solution path from simulation for Disc-Count=4		
Disc	Min-Disc	1-1-1-1-1-1-1-1
Disc	Min-Disc+1	-1---1---1---1-
Disc	Min-Disc+2	---1-------1---
Disc	Disc-Count	-------1-------
Peg	--1 (1)	----1---11----
Peg	-1- (2)	1--1-11-----1--
Peg	-11 (3)	-11-----11--1-11

Table 4-6 illustrates the transformation of the sequence associated with the disc and peg states from a three-disc simulation to a four-disc simulation. Note that the transformation pattern is identical duplicating the disc sequences with the new sequence introduced with a mid-point state

set and the copying peg sequences serially to combine pegs one and two to form target peg one sequence, three and one to form target peg two, and two and three to form target peg three.

Table 4-7: Transformation for New Hanoi Instance Using a Prior Instance (Count=4)

Attrib.	Value/Expression	Sequence/ Transform Results
T2 - Solution path for pattern for disc-count=3 to solution for disc-count=4 (S2->S3)		
E-Operation	Copy-Segment	1-11-11-1-
E-Operation	Insert-Bit	-1--1--1--
New-E-Operation	Binary-Expand	---------1
E-Source	Instance-1:Disc-1	1-1-------
E-Source	Instance-1:Disc-2	---1-1----
E-Source	Instance-1:Disc-3	------1-11
E-Source	Instance-1:Max-Steps (Power(2, Disc-Count) -1	---------1
E-Target	Instance-2:Disc-1	111-------
E-Target	Instance-2:Disc-2	---111----
E-Target	Instance-1:Disc-3	------111-
E-Target	Instance-2:Disc-4	---------1
E-Source	0	-1--1--1--
Attribute Operations		
A-Operation	Copy-Segment	1-11-11-1
A-Operation	Insert-Bit	-1--1--1-
A-Source	Instance-1:Peg-1	1--1-----
A-Source	Instance-1:Peg-2	--1--1---
A-Source	Instance-1:Peg-3	------1-1
A-Source	0	-1--1----
A-Source	1	-------1-
A-Target	Instance-1:Peg-1	111------
A-Target	Instance-2:Peg-2	---111---
A-Target	Instance-3:Peg-3	------111

Table 4-7 shows the sources, targets, and operations that represent the state sequences to generate the simulation solution for a disc count of four from an instance with a disc count of three.

Table 4-8: Sequence Comparison T1 to T2

Trans-form	Attrib.	Value/ Expression	Sequence
T1	E-Operation	Copy-Segment	1-11-1-
T2	**E-Operation**	**Copy-Segment**	**1-11-11-1-**
T1	E-Operation	Insert-Bit	-1-1---

T2	**E-Operation**	**Insert-Bit**	**-1-1-1----**
T1	New-E-Operation	Binary-Expand	------1
T2	***New-E-Operation***	***Binary-Expand***	***---------1***
T1	E-Source	Instance-S1:Disc-1	1-1----
T2	**E-Source**	**Instance-S1:Disc-1**	**1-1-------**
T1	E-Source	Instance-S1:Disc-2	---1-11
T2	**E-Source**	**Instance-S1:Disc-2**	**---1-1----**
T2	**E-Source**	**Instance-S2:Disc-3**	**------1-11**
T1	E-Source	Instance-S2:Max-Steps (Power(2, Disc-Count) -1	------1
T2	**E-Source**	**Instance-S3:Max-Steps (Power(2, Disc-Count) -1**	**---------1**
T1	E-Target	Instance-1:Disc-1	111----
T2	**E-Target**	**Instance-S3:Disc-1**	**111-------**
T1	E-Target	Instance-S2:Disc-2	---111-
T2	**E-Target**	**Instance-S3:Disc-2**	**---111----**
T1	E-Target	Instance-S2:Disc-3	------1---
T2	**E-Target**	**Instance-S3:Disc-3**	**------111-**
T2	**E-Target**	**Instance-S3:Disc-4**	**---------1**
T1	**E-Source**	**0**	-1--1—1
T2	**E-Source**	**0**	-1--1--1--1
Attribute Operations			
T1, T2	A-Operation	Copy-Segment	1-11-11-1
T1, T2	A-Operation	Insert-Bit	-1--1--1-
T1, T2	A-Operation	Insert-Bit	-1--1--1-
T1, T2	A-Source	Instance-1&2:Peg-1	1--1-----
T1, T2	A-Source	Instance-1&2:Peg-2	--1--1---
T1, T2	A-Source	Instance-1&2:Peg-3	------1-1
T1,T2	A-Source	0	-1--1----
T1,T2	A-Source	1	-------1-
T1	A-Target	Instance-1:Peg-1	111------
T1	A-Target	Instance-2:Peg-2	---111---
T2	A-Target	Instance-3:Peg-3	------111

In Table 4-8, the source sequences map from the transformation-solution instance one to transformation solution instance two. The sequences are targeting the problem of generating the sequence of operations needed to solve the Tower of Hanoi rather than the Tower of Hanoi itself. The transform column identifies the transform instance with the instance qualifier within the transform shown in the expression column of the chart. This is the first higher order problem transformation problem. Since there are no new attributes and the pattern for copying the sequences are the same, the attribute solution path is simply an exact duplication of the prior instance.

Table 4-9: Operation Transform Sequence (T1->T2)

TS1 - Solution path for generating sequence to transform from T1 to T2		
E-Operation	Prepend-Bit	111 111 --- 111 ---
E-Operation	New-Sequence	--- --- --- 1-- ---
E-Operation	Append-Bit	--- --- -11 --- ---
E-Operation	Copy-Segment	--- --- 1-- --- ---
E-Source	T1:Insert-Bit	--- 111 --- --- ---
E-Source	T1:Copy-Segment	111 --- --- --- ---
E-Source	T1:Binary-Expand	--- --- --- --- 111
E-Source	T1:Source-Entity	--- --- 1-- --- ---
E-Source	T1:New-Entity (Disc=3)	--- --- -11 1-- ---
E-Source	T2:Target-Entity	--- --- 1-- --- ---
E-Source	T2:New-Entity(Disc =Disc-Count)	--- --- --- 111 ---
E-Source	0	1-1 -1- --- 111 111
E-Source	1	-1- 1-1 -11 --- ---
Attribute Operations		
A-Operation	Copy-Segment	1-11-11-1
A-Source	Copy-Segment	1--1-----, --1--1---, ----- -1-1
A-Target	Copy-Segment	111------, ---111---, ----- -111

Legend:

Yellow: Copy-Segment alteration via bit-prepending
Green: Insert-Bit alteration via bit-prepending
Blue: Cloning of entities through segment copying
Olive: Creating of new entity sequence via bit-appending
Red: Offsetting binary expansion operation via bit-prepending

Table 4-9 examines the problem of generating the transformation sequence. This second-order transformation generates the sequences to convert the lower level transformation T1 to predict the sequences required for T2. TS1 generalizes all the source discs associated with T1 into a single operation and creates a new operation linked to the disc-count. Carrying out the operations in Table 4-9 transforms the source instance T1 to T2. For example in Table 4-8, T2's copy segment sequence requires an additional 101 at the start. The framework accomplishes this by referencing the copy-segment source along with a prepend-bit operator referencing 1, 0, 1 in steps one through three of the sequence as highlighted in yellow. The green highlight shows the sequences required to generate the T2 Insert-Bit operation sequence from the T1 Insert-Bit sequence operation that

also operate against the target entities but not the source. The blue highlight shows the copy-segment applied to all the entities of the source. The green highlight shows the operation sequences that transform the sequence for creating a new entity in T1 to create the new disc entity in T2. In T2, the binary expand operation is executed three steps later so prepend of bit zero is applied to that operation highlighted in red. Attribute operations are resolved as simply a duplication of the prior instance of the operation, source, and targets. The generation of a solution that transforms a transformation sequence set of operations from one instance to another moves up the abstraction level and gets closer to a generic solution for generating T-n+1 from T-n.

Table 4-10 shows an additional simulation instance with the following tables going through the same process for the earlier solution instances to derive a transformation sequence solution in Table 4-13. The attribute operations for the peg values are the same in these tables because TS1 was able to identify an exact copy operation for both T1 and T2 thus the generic solution is already in place for the attributes. The final higher order problem in Table 4-14 targets the higher order transformation-sequencing instance using the prior transformation-sequencing instance in order to generate a generic method for generating the transformation-sequencing instance. This then maps back to a problem attribute to define a repetition operation as well as an insertion operation based on this attribute. The insertion operation inserts a bit zero or bit one in the sequence depending upon the reference sequence/prediction target while the repetition operator (repeat-last-operation) repeats the insertion relative to the offset of the targeted instance, based on the disc-count from the source. The solution generation in Table 4-14 instance reflects the derivative operations of the lower level transforms to discover that the transform pattern between the two instances is simply a copy operation. Thus, no additional higher order transformations are necessary.

In this case, the disc count problem attribute becomes part of the sequence generation rule such that transformation-sequencing solutions can be generated for yet non-simulated problem instances. This capability allows the problem solver to predict the solution path incrementally for each lower level transformative property in a co-recursive fashion until the instance for the desired number of discs. In the last set of transformation operators, the prediction targets replace the original sources. This allows the solution to be repeatedly instantiated using its own predictions as the input until achieving the desired target instance.

The only capability needed for this is a reversal process that maps the relational state sequences to generate the actual state changes in the entity/attribute combinations from start to finish, performing an actual solution of an instance. Since the state capture process is complete in terms of all items necessary to reproduce a solution, a simple enumeration approach, covered in the next section, accomplishes this.

Table 4-10: State Sequences for Solved Simulations - Disc-Count = 4 and Disc-Count=5

S3 - Solution path from simulation for Disc-Count=4		
Attrib.	Value/Expression	Sequence/ Transform Results
Disc	Min-Disc	1-1-1-1-1-1-1-1
Disc	Min-Disc+1	-1---1--1---1-
Disc	Min-Disc+2	---1-------1---
Disc	Disc-Count	-------1-------
Peg	--1 (1)	----1----11----
Peg	-1- (2)	1--1-11-----1--
Peg	-11 (3) (3)	-11---11--1-11
S4 -Solution path from simulation for Disc-Count=5		
Disc	Min-Disc (1)	1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
Disc	Min-Disc+1 (2)	-1---1--1---1--1---1--1---1-
Disc	Min-Disc+2 (3)	---1-------1-------1-------1---
Disc	Min-Disc+3 (4	-------1-------------------1-------
Disc	Disc-Count (5)	---------------1---------------
Peg	--1 (1)	----1---11----1--1-11-----1--
Peg	-1- (2)	-11---11--1-11-----1----11----
Peg	-11 (3) (3)	1--1-11-----1--1-11---11--1-11

Table 4-11: Transformation for New Instance from Prior Instance (Count=5)

Attrib.	Value/Expression	Sequence/ Transform Results
T3 - Solution path for pattern for disc-count=4 to solution for disc-count=5 (S3->S4)		
E-Operation	Copy-Segment	1-11-11-11-1-
E-Operation	Insert-Bit	-1--1--1--1--
New-E-Operation	Binary-Expand	----------1
E-Source	Instance-1:Disc-1	1-1----------
E-Source	Instance-1:Disc-2	---1-1-------
E-Source	Instance-1:Disc-3	------1-1----
E-Source	Instance-1:Disc-4	---------1-1-
E-Source	Instance-1:Max-Steps (Power(2, Disc-Count) -1	----------1
E-Target	Instance-2:Disc-1	111----------
E-Target	Instance-2:Disc-2	---111-------
E-Target	Instance-1:Disc-3	------111----
E-Target	Instance-2:Disc-4	---------111-
E-Target	Instance-2:Disc-5	----------1
E-Source	0	-1--1--1--1--
Attribute Operations		
A-Operation	Copy-Segment	1-11-11-1
A-Operation	Insert-Bit	-1--1--1-
A-Source	Instance-1:Peg-1	1--1-----
A-Source	Instance-1:Peg-2	--1--1---
A-Source	Instance-1:Peg-3	------1-1
A-Source	0	-1--1----
A-Source	1	-------1-
A-Target	Instance-1:Peg-1	111------
A-Target	Instance-2:Peg-2	---111---
A-Target	Instance-3:Peg-3	------111

Table 4-11 shows the sources, targets, and operations that represent the state sequences to generate the simulation solution for a disc count of five from an instance with a disc count of four. This follows the identical pattern as for a transformation from three to four discs except the additional operations and the lengthening of the sequences.

Table 4-12: Entity Sequence Comparison from Transformation T2 to T3

Trans-form	Attrib.	Value/ Expression	Sequence
T2	E-Operation	Copy-Segment	1-11-11-1-

T3	**E-Operation**	**Copy-Segment**	**1-11-11-11-1-**
T2	E-Operation	Insert-Bit	-1-1-1---
T3	**E-Operation**	**Insert-Bit**	**-1-1-1-1----**
T2	New-E-Operation	Binary-Expand	---------1
T3	***New-E-Operation***	***Binary-Expand***	***----------1***
T2	E-Source	Instance-S3:Disc-1	1-1-------
T3	**E-Source**	**Instance-S3:Disc-1**	**1-1----------**
T2	E-Source	Instance-S3:Disc-2	---1-1---
T3	**E-Source**	**Instance-S3:Disc-2**	**---1-1----**
T2	E-Source	Instance-S3:Disc-3	------1-1---
T3	**E-Source**	**Instance-S3:Disc-3**	**------1-1----**
T3	**E-Source**	**Instance-S4:Disc-4**	**---------1-11**
T2	E-Source	Instance-S4:Max-Steps (Power(2, Disc-Count) -1	---------1
T3	**E-Source**	**Instance-S5:Max-Steps (Power(2, Disc-Count) -1**	**----------1**
T2	E-Target	Instance-1:Disc-1	111----
T3	**E-Target**	**Instance-S5:Disc-1**	**111-------**
T2	E-Target	Instance-S4:Disc-2	---111-
T3	**E-Target**	**Instance-S5:Disc-2**	**---111----**
T2	E-Target	Instance-S4:Disc-2	------111-
T3	**E-Target**	**Instance-S5:Disc-2**	**------111----**
T2	E-Target	Instance-S4:Disc-3	---------1---
T3	**E-Target**	**Instance-S5:Disc-3**	**---------111-**
T3	**E-Target**	**Instance-S5:Disc-4**	**----------1**
T2	**E-Source**	**0**	-1--1--1-1
T3	**E-Source**	**0**	-1--1--1--1–1

In Table 4-12, the source entity sequences map from the transformation solution instance two to transformation solution instance three. Again, the sequences are targeting the problem of generating the sequence of operations needed to solve the Tower of Hanoi and not the Tower of Hanoi itself. This is the second instance of the higher-order problem transformation problem. The next problem is to generate a sequence generation problem that can generate the sequence of instructions to create the transformation required to transform the simulation instances (TS1 ^ TS2 -> SG1). The sequence of operations for T3 is identical to T2; therefore, there is now a general solution by simply using the copy segment operation from the prior instance.

Table 4-13 –Sequence for Transforming a Transform Sequence (T1->T2 – Entities)

TS2 - Solution path for generating sequence to transform from T2 to T3		
E-Operation	Prepend-Bit	111 111 --- 111 ---
E-Operation	New-Sequence	--- --- --- 1-- ---
E-Operation	Append-Bit	--- --- -11 --- ---
E-Operation	Copy-Segment	--- --- 1-- --- ---
E-Source	T1:Insert-Bit	--- 111 --- --- ---
E-Source	T1:Copy-Segment	111 --- --- --- ---
E-Source	T1:Binary-Expand	--- --- --- --- 111
E-Source	T1:Source-Entity	--- --- 1-- --- ---
E-Source	T1:New-Entity (Disc=3)	--- --- -11 1-- ---
E-Source	T2:Target-Entity	--- --- 1-- --- ---
E-Source	T2:New-Entity(Disc =Disc-Count)	--- --- --- 111 ---
E-Source	0	1-1 -1- --- 111 111
E-Source	1	-1- 1-1 -11 --- ---
Attribute Operations		
A-Operation	Copy-Segment	1-11-11-1
A-Source	Copy-Segment	1--1-----, --1--1---, ----- -1-1
A-Target	Copy-Segment	111------, ---111---, ----- -111

Legend:

Yellow: Copy-Segment alteration via bit-prepending
Green: Insert-Bit alteration via bit-prepending
Blue: Cloning of entities through segment copying
Olive: Creating of new entity sequence via bit-appending
Red: Offsetting binary expansion operation via bit-prepending

Table 4-14: Sequence Generation for Generating Transform Solution

SG1 - Solution path for generating sequence to transform from TS2 to TS3		
E-Operation	Copy-Sequence	1
E-Source	TS(n+1)	1
E-Target	TS(n)	1

Table 4-14 meets the goal to generate a TS2 operational sequence from TS1 through a simple copy sequence. This ultimately provides the general solution to generate the solution directly without the need for simulation for further instances of Hanoi. This is because SG1 can generate the TS(n) Transform generation by copying TS(n-1) sequence. When reversing TS(n) from state sequences back to entity and attribute values, it generates Tn+1. When reversing T(n+1)

sequences, it generates entity and attribute values for simulation Sn+1. When S(n+1) is provided to T(n+1), the framework generates Sn+1. Generation of T(n+1) enables TSn+1 and the process repeats until S(n) meets the criteria for the number of discs. That is, if there is a request for the solution to a disc number of eight and UPRF has solved four instances in order to generate SG1, then simulation instances five through seven generate through the reversal process without any simulation in order to provide the criteria to generate the solution sequence for a simulation with eight discs.

4.2.4 Sequence Reversal

The prior section illustrated the process for transforming solution paths into higher order problems. In these higher order problems, the goal transitions to finding the technique for predicting the solution paths for the lower order problem for one instance from another instance – possibly multiple instances. This section will demonstrate how the state sequences transform into actual values in the database entities and attributes associated with the instance. This capability is necessary in order to generate a solution instance for a problem directly without the need for simulation. The purpose of this elaboration is to prove that the state capture associated with solution-path problem transformations is adequate to reproduce an actual solution.

In the final transformation, the disc count for the new problem instance is the reference variable. The framework simply needs to execute the problem setup, creating the initial instance in order to access this variable. In order to generate the targeted simulation, the framework must perform all of the transformations upon which the targeted simulation depends. This co-recursive process is the unwinding of the recursive problem solving process. As the framework performs

each higher order transformation, it generates the lower order solution instance until finally achieving the targeted simulation. This process is best illustrated by flipping the problem transformation process upside down and depicting the leading edge of the transformation generation associated with new instances as shown in Figure 4-9. In this process, the only required inputs are the predicted solution paths from the prior transformations. The source transformation instances are not necessary because the predicted instances are now the sources. In the diagram, the dashed connectors represent the inputs and the solid connectors represent the instances that will generate through the predictive transform operations.

Starting at the solution generator node, the process requires moving incrementally through each lower level instance until reaching the target instance for the problem variable. Therefore, even with this requirement to build out the number of instances incrementally, the actual time complexity increase is less than two times the complexity for a direct solution. There is also an overhead for building out the intermediate nodes, but this is a constant factor of three since the framework must perform only the leading edge of the transformations for each additional instance. Therefore, the number of operations is the number of operations in the target solution plus a constant factor for reversing from the solution generator back to the steps. Based on this, the time complexity of generating a solved instance is simply the addition of a linear constant in respect to the actual solution sequence. For example, it takes 31 steps to solve a Tower of Hanoi problem with five discs ($2^5 - 1$) using the most efficient set of moves. The solution generator is able simply to copy the sequences from the prior instance to generate the transformation sequence, which then generates the simulation sequences.

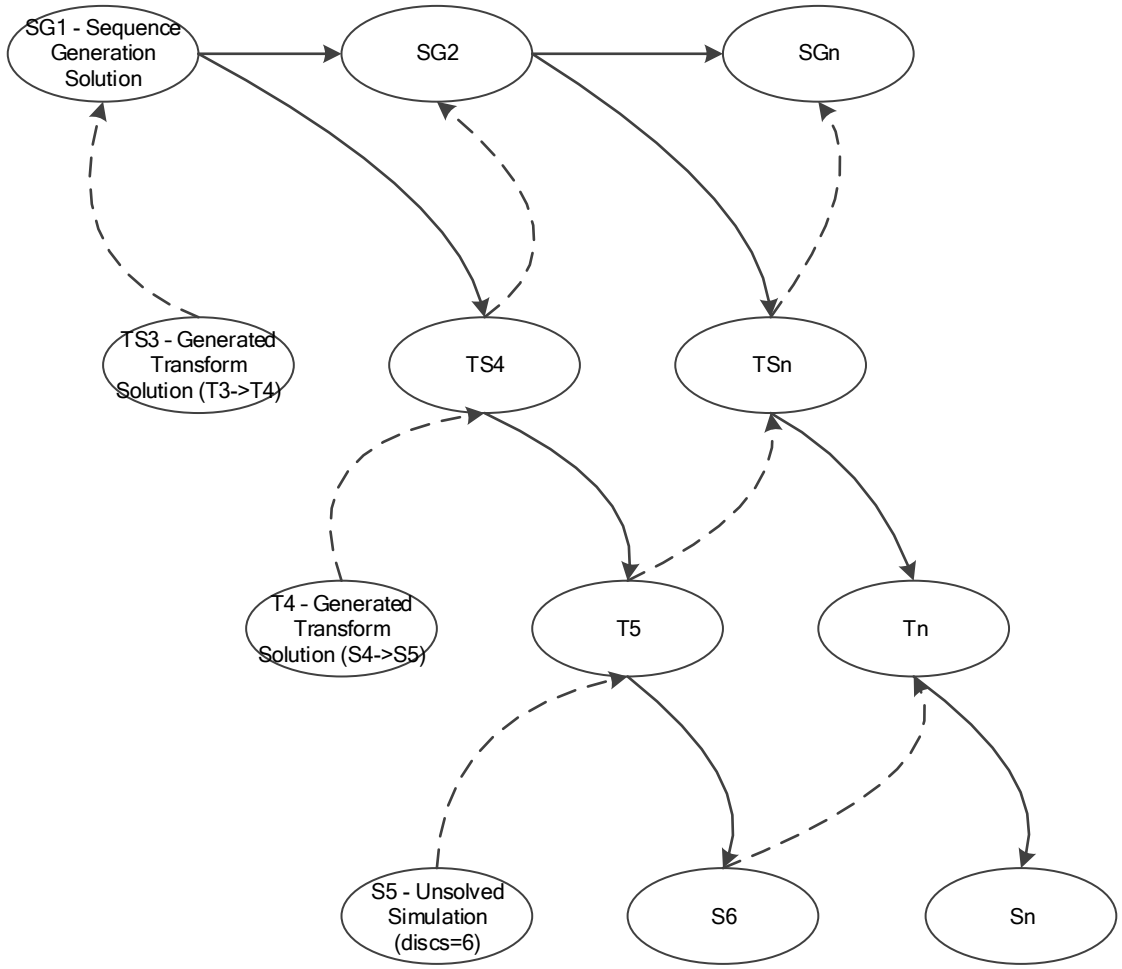


Figure 4-9: Transformative Reversal Process

The reversal process starts at SG1, which utilizes TS3 as the input to generate TS4. TS4 uses the prior predicted instance of T4 as the input in order to generate T5. T5 then uses the last simulation S5 as the input to generate the prediction solution sequence for S6. This process is repeatable for incrementally increasing the number of instances until solving the target instance. Table 4-17 illustrates how the output of the sequence states are transformed into attribute value sequences that represent the specific state changes. State sequences that define the entity and attribute values intersect in order to define the specific values for the entity attribute combinations.

The framework must examine the sequence reversal process starting at the highest order and working backwards since only the highest order transformation is able to create the dependent instance. This dependent instance is necessary to generate the solution path tree for a new solution instance to the simulation problem. For example, given S5 as the last simulation instance, only T5 can create S6 and only TS4, which does not exist, can create T5. However, SG1 can create TS4 by using the output of SG1 as the input for the next instance of SG1 as defined by the Replace-Source operation in the final transforms of the SG1 instance. Once the framework creates TS4, it can then generate the required instances to predict S6.

Although the reversal process must start at the highest order transformation level, the process is easier to understand at the lower level transformation so before embarking on the full process, let us examine a lower level process. Table 4-17 shows how a sequence is implemented into the value sequence data table from the activation sequences in Table 4-16. The value sequence is query-able to define the exact solution steps because it identifies the specific attribute value to assign to an attribute value for a specific entity instance over a sequence range. If the value sequence populates accurately from the state activation sequences, then it is easy to replicate the exact solution to a problem instance.

Table 4-15: State Sequences for S1

Instance	Ref.	Attrib.	Binary Value/Expression	Sequence/ Transform Results
S1 - Solution path from simulation for Disc-Count=2				
1	1	Disc	001 (1)	1-1
1	2	Disc	010 (2)	-1-
1	3	Peg	001 (1)	---
1	4	Peg	010 (2)	1--
1	5	Peg	011 (3)	-11

Table 4-16: Translating attribute sequences to unique entity/attribute values

Value Sequences for S1					
Entity Value	Attribute Value	Start Step	End Step	Source Seq. Refs	Source Sequence Intersection
Disc=1	Peg=1			1	
Disc=1	Peg=2	1	2	1,4	1-1,1--
Disc=1	Peg=3	3	3	1,5	1-1,-11
Disc=2	Peg=1			2	
Disc=2	Peg=2			2	
Disc=3	Peg=3	2	3	2,5	-1-, -1-

An intersection of an entity with a specific value, an attribute with a specific value, and a sequence must join in order to define a definitive value to assign to an entity for an attribute at any given point. For example, the entity value Disc=1 is supported by the activation sequence that shows disc one is active at step one and step three, but Peg=1 is not active at either of these steps. Therefore no causative sequence intersection exists to indicate a value for Disc=1 and Peg=1 for any of the steps. However, Peg=2 is activated at step one as well as Disc=1, therefore an intersection exists resulting in a value sequence active at step one. Value sequences retain their values throughout the sequence until a change occurs. Since Disc=1 does not have an activation on step two, its sequence remains intact for the current value. Thus the value sequence for Disc=1, Peg=1 is true from step one to step two. On step three, Disc=1 is once again activated, but the activated attribute is Peg=3. Therefore the value sequence for Disc=1/Peg=2 terminates and a new value sequence starts with Disc=1/Peg=3. The framework can easily query these value sequences to display the move sequence by displaying all of the points where the value sequence changes as illustrated in Figure Table 4-17.

For the transformation to be valid, the higher-level activation sequences must be able to generate the sequence of operations used to attain the predictive sequence at each lower level. The

process for generating the transformation solution-sequence paths is the same as that for the base simulation instance. Table 4-18 shows the transformation sequence for the problem T1 along with the steps at which the attribute is set to the associated value based on the sequence. The reversibility mechanism in this instance needs to generate the steps utilized in T2 from the transform operators. The prior section already demonstrated that the transformation operators are able to produce the predicted solutions. At the higher-level transformation, it is important to show that the sequences and usages which will be used for the next higher order transformation are correct for replicating the transformation operations against the lower level activation sequences of the base simulation. Table 4-19 shows how the framework reverses state sequences to replicate the operations at the specific steps in the solution.

Table 4-17: Transform Sequence Steps

Attrib.	Value/Expression	Sequence/ Transform Results	Steps
T1 - Solution path for pattern for disc-count=2 to solution for disc-count=3 (S1->S2)			
E-Operation	Copy-Segment	1-11-1-	1,3-4,6
E-Operation	Insert-Bit	-1--1--	2,5
New-E-Operation	Binary-Expand	------1	7
E-Source	Instance-1:Disc-1	1-1----	1,3
E-Source	Instance-1:Disc-2	---1-11	4,6,7
E-Source	Instance-1:Max-Steps (Power(2, Disc-Count) -1	------1	7
E-Source	0	-1--1--	2,5
E-Target	Instance-2:Disc-1	111----	1-3
E-Target	Instance-2:Disc-2	---111-	4-6
E-Target	Instance-2:Disc-3	------1	7
Attribute Operations			
A-Operation	Copy-Segment	1-11-11-1	**1,3-4,6-7,9**
A-Operation	Insert-Bit	-1--1--1-	2,5,8
A-Source	Instance-1:Peg-1	1--1-----	1,4
A-Source	Instance-1:Peg-2	--1--1---	3,6

A-Source	Instance-1:Peg-3	------1-1	2,5
A-Source	0	-1--1----	2,5
A-Source	1	-------1-	8
A-Target	Instance-1:Peg-1	111------	1-3
A-Target	Instance-2:Peg-2	---111---	4-6
A-Target	Instance-3:Peg-3	------111	7-9

Table 4-18: Reversibility Matrix for Transform Sequence

Entity/Attribute.	Value	Start Step	End Step
E-Operation	Copy-Segment	1	1
		3	4
		6	6
	Insert-Bit	2	2
		5	5
New-E-Operation	Binary-Expand	7	7
E-Source	Instance-1:Disc-1	1	1
		3	3
	Instance-1:Disc-2	4	4
		6	7
	Instance-1:Max-Steps (Power(2, Disc-Count) -1	7	7
	0	2	2
		5	5
E-Target	Instance-2:Disc-1	1	3
	Instance-2:Disc-2	4	6
	Instance-2:Disc-3	7	7
A-Operation	Copy-Segment	1	1
		3	4
		6	7
		9	9
	Insert-Bit	2	2
		5	5
		8	8
A-Source	Instance-1:Peg-1	1	1
		4	4
	Instance-1:Peg-2	2	2
		5	5
	Instance-1:Peg-3	3	3
		6	6
	0	2	2
		5	5
	1	8	8
A-Target	Instance-1:Peg-1	1	3
	Instance-2:Peg-2	4	6
	Instance-3:Peg-3	7	9

In Table 4-19, the attributes are stored in multiple entities each uniquely identified by the problem step where step represents a discrete state in the solution sequence. The framework defines each sequence entity in terms of the attribute that stores a pointer to the value sequence from the lower level problem. In the first-level transformation, the values reflect the sequences from the simulation instances. As the problem progresses to higher order transformations, the source values point to the underlying sequences for the transformation problems themselves.

These activation sequences are reversible to generate the value sequences that reflect the required transformations. The framework understands this process in terms of TS1, which predicts T2. The value sequences from the TS1 instance map to the attributes for the higher order problem, which follow the same pattern as T1. Since TS1 can generate T2 successfully in the same format at T1 and T1 generates S2 successfully, then T2 will generate S3 successfully. It now only remains for SG1 to generate TS2 successfully. Since SG1 can generate TS2 value sequences that implement the T3 transform successfully, SG1 is correct for generating a new instance of TS. This new instance cascades down to new solved instances of S because T1 was reversible and all higher order transformations follow the same model of referencing the lower level transformation sequences. Therefore, the following recursive/co-recursive sequence exists:

Recursive Discovery:

S1, S2, S1 ^ S2->T1

S3, S2 ^ S3->T2

T1 ^ T2->TS1

S4, S3 ^ S4 -> T3

T2 ^ T3 -> TS2

TS1 ^ TS2 -> SG1

Recursivity Pivot Point:

TS2 ^ SG -> TS3

Co-recursive Cascade

T3 ^ TS3 -> T4

S4 ^ T4 -> S5

TS3 ^ SG -> TS4

T4 ^ TS4 - > T5

S5 ^ T5 -> S6

Co-recursive Relation:

TS(n) ^ SG -> TS(n+1)

T(n) ^ TS(n+1) -> T(n+1)

S(n) ^ T(n+1) -> S(n+1)

4.3 Universal Problem Representation

In the prior section, the Tower of Hanoi was modeled to show the UPRF process for generating solution simulations, transforming solution sequences as the goal state for a higher order problem, and generating the solution sequences associated to each level of higher order problem (until reaching an equilibrium point where the sequence was generic regardless of the instance). At that point, the process was reversible to generate the sequences back into the database to model the steps accurately. This section focuses on the problem of problem solving itself. The earlier section

made the assertion that the same process utilized to discover solution patterns via simulative search could search for pattern sequences by testing transform operators. This section provides the problem definition of the universal optimization problem that is associated with the sequence discovery and is fully recursive and co-recursive.

Figure 4-8 shows the schema for UPRF sequence discovery problem or the universal discovery problem (UDP). The goal of the problem is to discover the operators that generate the targeted sequence. This aligns with the prior section wherein the solution state sequence of each entity/attribute combination became the target of a higher order problem. UDP utilizes the same schema as the base case problem including Hanoi. This means that UDP fits into the same framework for simulation as any other problem and helps validate that the problem definition language (UPDL) is robust enough to handle complex problems. In the next chapter on practicality, further examples help validate the extensibility of UPDL as well as the framework (UPRF) to model and solve a variety of problems.

A distinguishing aspect of the UDP from other problems is that the problem is that the simulation relies on a special operator ("TEST_TRANSFORM") to actually generate the sequences from the operators and test them. However, this aspect does not prevent UDP from operating within the self-same framework and it meets the criteria for the continuous improvement model by exposing all the attributes, expressions, and queries back to the framework that are utilized to meet the goal defined by the problem in the schema.

A second distinguishing aspect is the use of the "SELECT_ INTERNAL". This provides a means for the UDP to query objects within the database without the need for them to replicate to the standard entity/attribute tables. This ensures access to the lower level problem data without the

need to replicate this back into the structure. The difference in implementation does not compromise the rule for the system requiring all operations to be traceable back to problem states so long as views insulate the underlying implementation and provide uniform representation of problems regardless of height in the solving hierarchy. The actual implementation of the learning solution is done through the standard entities/attributes persisted back into the database. This data is accessible then through the higher order transformations without any limit, since every instantiation of the UDP utilizes the same definition. Triggering a UDP problem occurs when solution of more than one instance at the same level in a problem. For example in Hanoi, solving a three-disc simulation after solving a two-disc simulation generates a UDP transformation problem whose goal is to discover the sequence of operations to generate a three-disc solution from a two-disc solution using the relational state sequences.

Figure 4-6 depicts the UDP definition. The definition obfuscates much of the delineation from the prior operational proof through highly encompassing functions for testing the transformations. The definition could provide more detail to support lower granularity, but there is little value in this since the transformation operation generates the information needed for traceability back to the correlations associated with the transform operator used and the sequences. The goal of the UDP problem is simply to execute a broad query, which filters using the transform testing function that then returns all valid instances of source and target entities along while iterating through each transform operator. This forms the output of the query. When the full sequence generates successfully for all sequences, the process is complete. At this point, the transformation sequence forms the first instance of the transformation problem. When UPRF finishes the simulation for a third lower level instance such as for a disc count of five, then the framework will detect an

untransformed instance. It will then attempt to find the method for generating the transform sequence to transform the transform operations performed on the simulation for three discs to achieve operations to solve four discs in order to generate the operations needed to transform the simulation for four discs to solve five discs. This becomes a third-level problem when encountering more simulations until reaching generalization as described earlier.

```
<Upr xmlns:xsi="ProblemDef-v8.xsd" Version="11.0">
  - <Problem TransitionQuery="Test-Transformation-Operator" GoalQuery="Test-All-Instances-
    Predicted" DeriveFrom="Get-Untransformed-Instance" Id="Universal-Discovery">
      - <Attribute Id="Transformation-Operator-List">
            <Expression>Get-Transformation-Operator</Expression>
        </Attribute>
      - <Attribute Id="Get-Source-Instance">
            <Expression>Get-UnTransformed-Source-Instance</Expression>
                <!--An untransformed instance is an instance associated to a problem that has another
                instance from the same problem but has not yet generated a higher order problem mapped to
                the lower instance. This attribute contains the source instance to use for predicting the target
                instance-->
        </Attribute>
      - <Attribute Id="LearningInstanceId">
            <Expression>SYSTEM_CURRENT_PROBLEMID</Expression>
                <!-- Gets the Base Instance ID of this problem instance A new problem is generated for each
                higher-order transform that includes all of the instances reflecting-->
        </Attribute>
      - <DataEntity DeriveFrom="SYSTEM_STEPNUMBER" Id="Predicted-Value-Sequence">
            <Attribute Id="Operator-Id" Domain="Get-Transformation-Operator"/>
            <Attribute Id="Source-Value-Sequence"/>
        </DataEntity>
      - <Expression Id="Get-Transform-Operator" Operator="SELECT_INTERNAL">
            <Argument Parameter="Table" Attribute="Ref.TransformationOperator"/>
            <Argument Parameter="RETURN_ATTRIBUTE" Attribute="TransformationOperatorId"/>
        </Expression>
      - <Expression Id="Get-Value-Sequence" Operator="SELECT_INTERNAL">
            <Argument Parameter="Table" Attribute="Engine.view_ValueChangeSequence"/>
            <Argument Parameter="RETURN_ATTRIBUTE" Attribute="ValueSequenceId"/>
            <Argument Parameter="FILTER_ATTRIBUTE" Attribute="ProblemInstanceId"/>
            <Argument Parameter="FILTER_ATTRIBUTE" Attribute="TransformOperatorId"/>
                <!-- This returns sequences for an instance-->
        </Expression>
      - <Expression Id="Get-Untransformed-Problem-Id" Operator="SELECT_INTERNAL">
            <Argument Parameter="RETURN_ATTRIBUTE" Attribute="ProblemId"/>
            <Argument Parameter="FILTER_ATTRIBUTE" Attribute="ProblemInstanceId"/>
        </Expression>
      - <Expression Id="Get-Untransformed-Instance-StepTotal" Operator="SELECT_INTERNAL">
            <Argument Parameter="Table" Attribute="Engine.ProblemInstance"/>
            <Argument Parameter="RETURN_ATTRIBUTE" Attribute="StepTotal"/>
            <Argument Parameter="FILTER_ATTRIBUTE" Attribute="ProblemInstanceId"/>
                <!-- TransformInstanceStatus of 1 indicates this is a transform-capable instance (2nd or higher
                instance) and the learning instance has yet not been solved)-->
        </Expression>
      - <Expression Id="Get-Untransformed-Instance" Operator="SELECT_INTERNAL">
            <Argument Parameter="Table" Attribute="Engine.ProblemInstance"/>
            <Argument Parameter="RETURN_ATTRIBUTE" Attribute="ProblemInstanceId"/>
            <Argument Parameter="FILTER_ATTRIBUTE" Attribute="TransformInstanceStatus"/>
                <!-- TransformInstanceStatus of 1 indicates this is a transform-capable instance (2nd or higher
                instance) and the learning instance has yet not been solved)-->
        </Expression>
      - <Expression Id="Get-Untransformed-Source-Instance" Operator="SELECT_INTERNAL">
            <Argument Parameter="Table" Attribute="Engine.ProblemInstance"/>
            <Argument Parameter="RETURN_ATTRIBUTE" Attribute="ProblemInstanceId"/>
            <Argument Parameter="SORT_ATTRIBUTE" Attribute="ProblemInstanceId"/>
            <Argument Parameter="FILTER_ATTRIBUTE" Attribute="ProblemId"/>
            <Argument Parameter="SORT_DIRECTION" Attribute="ASC"/>
            <Argument Parameter="ROWS_TO_RETURN" Attribute="1"/>
                <!-- This expression assures the lower level instance is selected -->
        </Expression>
```

```
        <!-- This expression assures the lower level instance is selected -->
    </Expression>
        <!-- Keep solution paths alive as long as successful solution still selected from prior rules
        generation -->
        <!-- Allow failures to be in the selection set and keep each solution branch alive unless a successful
        path is removed. The final goal is to determine the solution branch that keeps all successful
        instances and prunes out all failure instances in the least amount of steps-->
        <!-- Seelection State = -1 for failed select; 0 means not yet determined (initial state) -->
  - <Query Id="Predict-Solution">
        <Extract Id="Problem-Instance-Id" Expression="Universal-Discovery"/>
      - <Extract Id="Problem-Id" Expression="Get-Untransformed-Problem-Id">
            <Criteria Attribute="ProblemInstanceId" Extract="Problem-Instance-Id"
                CompareOperator="EQUAL"/>
        </Extract>
            <!-- Universal Discovery -->
        <Extract Id="Transform-Operator" Expression="Get-Transform-Operator"
            OutputAttribute="Transform-Operator"> </Extract>
      - <Extract Id="Untransformed-Target-Instance" Expression="Get-Untransformed-Instance"
        JunctionType="AND">
            <Criteria Attribute="ProblemId" Extract="Problem-Id" CompareOperator="EQUAL"/>
            <Criteria Attribute="TransformInstanceStatus" Extract="1" CompareOperator="EQUAL"/>
        </Extract>
      - <Extract Id="Untransformed-Source-Instance" Expression="Get-Untransformed-Source-
        Instance" JunctionType="AND">
            <Criteria Attribute="ProblemId" Extract="Problem-Instance-Id"
                CompareOperator="EQUAL"/>
            <Criteria Attribute="ProblemInstanceId" Extract="Untransformed-Instance"
                CompareOperator="LESS_THAN"/>
        </Extract>
      - <Extract Id="Value-Sequence" Expression="Get-Value-Sequence" OutputAttribute="Source-
        Value-Sequence" JunctionType="OR">
            <Criteria Attribute="ProblemInstanceId" Extract="Untransformed-Target-Instance"
                CompareOperator="EQUAL"/>
            <Criteria Attribute="ProblemInstanceId" Extract="Untransformed-Source-Instance"
                CompareOperator="EQUAL"/>
                <!--Get all of the value sequences associated with the Source or Target instance -->
        </Extract>
      - <Extract Id="Predicted-Value-Sequence" Expression="Get-Value-Sequence"
        JunctionType="AND">
            <Criteria Attribute="ProblemInstanceId" Extract="Problem-Instance-Id"
                CompareOperator="EQUAL" StepNumber="0"/>
            <Criteria Attribute="TransformOperatorId" Extract="Transform-Operator"
                CompareOperator="EQUAL"/>
                <!-- Gets the sequences generated by the transform operator for this instance so they can
                be compared-->
        </Extract>
        <Filter Id="Test-Transform" Operator="TEST_TRANSFORM_CUMULATIVE" Extract2="Value-
            Sequence" Extract1="Predicted-Value-Sequence"/>
            <!-- The only way a result is returned is if the test transform succeeds and the prediction
            sequence matches the value sequence for all bits generated. The sequence may still be not-
            complete. No fail query is needed because there candidate sequences are only selected if they
            have matched cummulatively-->
    </Query>
  - <Query Id="Test-All-Steps-Predicted">
        <Extract Id="Current-Step" Expression="SYSTEM_STEPNUMBER" OutputAttribute="1">
            </Extract>
        <Extract Id="Target-Instance-Step-Count" Expression="Get-Untransformed-Instance-
            StepTotal"> </Extract>
        <Filter Id="Check-All-Predictions-Correct" Operator="TEST_TRANSFORM_ALL"
            Extract2="Value-Sequence" Extract1="Predicted-Value-Sequence"/>
    </Query>
        <!-- Process is done whe all of the instances have been predicted correctly for failure or success
        -->
:/Problem>
```

Figure 4-10: Universal Discovery Problem

4.4 Chapter Summary

This chapter provided an operational proof for the feasibility of a universal problem resolution framework (UPRF). It enumerated an inductive proof showing how the proposed UPRF design met the key requirements for generic problem definition, solution space probing, and higher-order transformation. The chapter concluded with a proof-by-example using the Tower of Hanoi in which the solution path, through simulation and transformation to higher levels of abstraction, evolves from brute-force exploration to generating a generic transformation that reduces identification of the solution path of further instances of the puzzle to a linear process.

5. PRACTICALITY

To this point, the dissertation has focused on the use of Hanoi to demonstrate UPRF. The Tower of Hanoi is an NP-hard problem, but it has commonality with virtually all types of computational problems; it possesses an initial state, manifests in multiple instances, and has a generalized solution adaptable for any instance to meet a goal state. Wide varieties of problems share these attributes including NP-complete problems such as the traveling salesman problem (TSP). The TSP also has an initial state, goal state and is configurable as different instances. General solutions exist that require exponential complexity as well as approximation solutions that attempt to optimize the solution discovery at the risk of not being perfect. A wide variety of problems fall into the NP-complete domain and every NP problem is reducible to another NP problem. The problem of discovering an algorithm is actually an NP-complete problem. To be practical, UPRF must be able to leverage current technology or at least the technology being developed to solve problems in practical time limits within reasonable resource constraints.

Discussions around practicality involve the concept of scalability. This chapter examines scalability in terms of both UPRF functionality across problem types as well as in terms of implementation of the system within technological constraints. This chapter provides answers to the following questions:

1. **Scalability for other problems**: How does UPRF handle other problems including non-deterministic or probabilistic scenarios? Is it able to scale for problems that are more

complex or for different types of problems? Does the UPRF sequence recognition model scale to higher-order complexity for the base Tower of Hanoi scenario when the number of pegs is a variable as well as the number of discs? Can it optimize solutions for NP-complete problems including finding an even larger TSP route than currently possible through approximation optimizations? Can UPRF support collaborative or cooperative multi-agent scenarios? For example, could it continually play itself in a game such as Chess and improve itself without input from a SME?

2. **Implementation Feasibility:** Is UPRF practical for implementation? How can it scale to leverage technology innovations?

5.1 Scalability for Other Problems

Scalability includes supporting expanding capacity without the need for extensive modification of the system. The schematization model of UPRF supports this by providing a generic representation of any relational state through functions. UPRF supports external referencing as well as self-referencing through comprehensive polymorphism data structures. In UPRF, a problem may manifest itself as an entity that another problem can reference. Items within the data structure are also bi-directional within the context of constraints that prevent endless reference loops while using dependency detection to iterate through scenarios in dependency sequence. In this chapter, UPRF uses this capability to support competitive and collaborative scenarios including an example from Tic-Tac-Toe and discussions of Chinese checkers and Canasta. The Tic-Tac-Toe example delves into the variations imposed by multi-agent scenarios and iterates through a solution scenario from start to finish for an application of symmetry to replicate a solution path. This chapter discusses feasibility for the implementation of Chinese

checkers and Canasta scenarios based on the multi-agent paradigm established in the Tic-Tac-Toe example.

An additional area of scalability provided by UPRF concerns the types of solutions that are pursuable. The Tower of Hanoi illustrated the pursuit of a definitive deterministic goal; the goal specified an exact solution to the problem. However, because the goal state derives from functions against the relational state and such functions may be aggregate in nature rather than exact, UPRF can work towards solutions that are not exact and based on an energy function. In this chapter, the zero-sum subset problem, a manifestation of the NP-complete problem type, serves as an example. NP Problems include scenarios such as the TSP. This example shows all such problems are reducible to each other and the subset problem is easy to represent in queries. In the zero subset problem, the goal defined involves minimizing steps rather than finding the solution in a specific number of steps. This allows UPRF to attempt solutions to NP-complete problems and utilize the transformational learning process to identify optimizations toward the problem. While the dissertation does not attempt to show the reduction through successive optimizations to polynomial time of an NP-complete problem, it does show that the maximum degree of optimization is a natural outcome of the UPRF approach.

Another example of scalability related to problem types are probabilistic scenarios. This chapter provides an example of a probabilistic scenario for optimizing trades in the stock market. In this scenario, UPRF attempts to find the optimal method for making profitable trades with the transformations involving different instances of time to identify repeatable patterns. Some research asserts the stock market is an efficient system and therefore not providing any repeatable optimization technique that is better than chance. This dissertation does not attempt to prove this

either way, but it does show how UPRF can help to test such a hypothesis through the simulation and transformation process.

5.1.1 Increasing Complexity – K-Peg Tower of Hanoi

In the k-peg Tower of Hanoi, the number of pegs themselves become a variable rather than fixed at three pegs only. This section explores the background of the k-peg Hanoi, models a k-peg problem, utilizes simulation to generate the sequences, examines the patterns that emerge from this exercise, and analyzes how the UPRF process ultimately identifies the transformation sequence to extend the 3-peg solution to any number of pegs. Although this section does not go into the degree of depth as Chapter 3 did regarding the transformation sequence recognition process, it does provide proof from inspection that the same technique used to derive the general formula for 3-peg, k-disc variations will yield a general formula aligning to Reve's [120] and solving the puzzle in the minimum steps. The number of optimal steps for different variations with pegs and discs is a complex formula [121]. UPRF does not need to know the optimal number in order to pursue a simulative solution, as it is able to exhaust all possible paths until finding the minimum. Providing the number of steps is an optimization to reduce the amount of simulation work.

Debate regarding the proof for an optimal solution approach for k-peg Hanoi persists but brute-force simulation achieves the minimum number of steps for scenarios where k is sufficiently small. Using the same approach as with the standard Tower of Hanoi, UPRF is able to model the problem very simply for presentation to the simulator (Figure 5-1). This example illustrates an alternative definition method using standard Structured Query Language (SQL) to define views and functions based on views from the generic UPRF schema. Figures 5-2 and 5-3 contain the

SQL code for modelling the detailed K-peg tower problem and then returning the rows associated with the next possible moves at any point in the simulation process.

```
- <Problem Id="HanoiNpeg">
        <!--Problem Control Attributes Iterate through n peg/n disc with offset variable to affect
        what to use for the disc This allows multiple simulations to be correlated with different
        discs as well as increasing the peg through a single attribute change. -->
    <Attribute Id="Hanoi_Disc_Count"/>
        <!-- Is also the peg count -->
    <Attribute Id="Hanoi_Min_Disc_Size" Value="1"/>
    <Attribute Id="Hanoi_Min_Disc_Count" Value="3"/>
    <Attribute Id="Hanoi_Max_Disc_Count" Value="5"/>
    <Attribute Id="Hanoi_Disc_Offset" Value="0"/>
        <!-- Offset to apply to set the number of discs higher than pegs i.e. 6 discs/4 pegs offset
        would be 2-->
  - <Entity Id="Hanoi_Disc" Key="Disc_Size">
        <Attribute Id="Disc_Size" IsStatic="true"/>
        <Attribute Id="Disc_Peg" Value="1"/>
    </Entity>
        <!--Initial State Definition-->
        <!--The first Expression for the initial game state effectively generates a problem for
        each variation on the number of discs within the range. The next Expression is evaluated
        in the context of all of the spawned problems and then applied to the discs for the
        respective problem. -->
  - <State Id="Hanoi_Initial_State" Rule="ProblemStartRule" JoinOp="AND">
        <Expression Id="Hanoi_Game_Start" Sequence="1"
            Operator="HanoiNPeg.tdf_SetupProblem"> </Expression>
    </State>
  - <State Id="Hanoi_Disc_Initial_State" Rule="EntityStartRule" JoinOp="And">
      - <Expression Id="Hanoi_Disc_Start" Sequence="2"
        Operator="HanoiNPeg.tdf_SetupDisc">
            <AttributeMap Target="Disc_Size" Source="Hanoi_Disc_Count">
                </AttributeMap>
        </Expression>
    </State>
        <!--Transitive State Definition In the Transitive mode, each instance of the problem has
        a different problem ID The new peg value along with the disc to select for update is
        returned for each problem. This result set may contain multiple entries for different
        possibilities. To support this, the problem is branched off to a new problem for each of
        the possibilities. -->
  - <State Id="Hanoi_Transition_State" Rule="UpdateRule" JoinOp="AND">
        <Expression Id="Hanoi_Selectable_Disc_And_Peg"
            Operator="HanoiNPeg.tdf_GetNextMove"> </Expression>
    </State>
        <!--Goal State Definition For Goal type State, the funtion is to return 1 if reached, 0 if
        not reached. -->
  - <State Id="Hanoi_Final_State" Rule="GoalTest" JoinOp="AND">
        <Expression Id="Hanoi_Check_Goal"
            Operator="HanoiNPeg.tdf_CheckAllOnPegN"> </Expression>
    </State>
        <!--A view is generated for all problems in terms of the relational structure of the
        problem-->
  + <Operator Id="HanoiNPeg.tdf_GetNextMove" Result="Table">
  + <Operator Id="HanoiNPeg.tdf_CheckAllOnPegN" Result="Scalar">
  + <Operator Id="HanoiNPeg.tdf_SetupProblem" Result="Table">
  + <Operator Id="HanoiNPeg.tdf_SetupDisc" Result="Table">
  </Problem>
```

Figure 5-1: Hanoi K-Peg Definition

```
CREATE VIEW [HanoiNpeg].[view_Problem_Detail]
as
SELECT  Hanoi_Entity.ItemId as ProblemId,
        Hanoi_Entity.CurrentStepNumber AS System_StepNumber,
        Hanoi_Disc_Count.NumericValue as Hanoi_Disc_Count,
        Hanoi_Disc_Offset.NumericValue as Hanoi_Disc_Offset,

        Hanoi_Disc_Entity.ItemId AS Hanoi_Disc_EntityId,

        Disc_Size_LastValue.NumericValue as Disc_Size_LastValue,
        Disc_Peg_LastValue.NumericValue as Disc_Peg_LastValue,
        Disc_Size.NumericValue as Disc_Size,
        Disc_Peg.NumericValue as Disc_Peg,
        Disc_Size_LastValue.ItemId as Disc_Size_LastValue_Id,
        Disc_Peg_LastValue.ItemId as Disc_Peg_LastValue_Id,
        Disc_Size.ItemId as Disc_Size_Id,
        Disc_Peg.ItemId as Disc_Peg_Id,
        Hanoi_Entity.ProblemSolutionState AS SolutionState
-- Entities
FROM Data.view_Problem_Item Hanoi_Entity
INNER JOIN Data.view_Entity_Item Hanoi_Disc_Entity
    ON  Hanoi_Disc_Entity.ProblemId = Hanoi_Entity.ItemId
    AND Hanoi_Disc_Entity.ItemName = 'Hanoi_Disc'
    AND Hanoi_Disc_Entity.ItemId <> Hanoi_Entity.ItemId
-- Static Attributes (IsStatic = 1 on Attribute)
INNER JOIN Data.view_Attribute_Item Hanoi_Disc_Count
    ON   Hanoi_Disc_Count.EntityId = Hanoi_Entity.ItemId
    AND  Hanoi_Disc_Count.ItemName = 'Hanoi_Disc_Count'
INNER JOIN Data.view_Attribute_Item Hanoi_Disc_Offset
    ON   Hanoi_Disc_Offset.EntityId = Hanoi_Entity.ItemId
    AND  Hanoi_Disc_Offset.ItemName = 'Hanoi_Disc_Offset'
-- Generated attributes for Last Selected and Last Updated values
LEFT JOIN Data.view_Attribute_Item Disc_Size_LastValue
    ON   Disc_Size_LastValue.EntityId = Hanoi_Entity.ItemId
    AND  Disc_Size_LastValue.ItemName = 'Disc_Size_LastValue'
LEFT JOIN Data.view_Attribute_Item Disc_Peg_LastValue
    ON   Disc_Peg_LastValue.EntityId = Hanoi_Entity.ItemId
    AND  Disc_Peg_LastValue.ItemName = 'Disc_Peg_LastValue'
INNER JOIN Data.view_Attribute_Item Disc_Size
    ON  Disc_Size.EntityId = Hanoi_Disc_Entity.ItemId
    AND Disc_Size.ItemName = 'Disc_Size'
INNER JOIN Data.view_Attribute_Item Disc_Peg
    ON  Disc_Peg.EntityId = Hanoi_Disc_Entity.ItemId
    AND Disc_Peg.ItemName = 'Disc_Peg'
WHERE Hanoi_Entity.ItemName = 'HanoiNPeg'
```

Figure 5-2: Hanoi K-Peg SQL View based on UPRF Schema

```
CREATE FUNCTION [HanoiNpeg].[tdf_GetNextMove]
(
    @ProblemId MONEY
)
RETURNS TABLE
AS
RETURN
    SELECT h.Disc_Peg_Id AS Id, c.NumericValue as Value
    FROM Engine.NumericConstant c
    INNER JOIN HanoiNPeg.view_Problem_Detail h
        ON h.ProblemId = @ProblemId
    INNER JOIN HanoiNpeg.MaxMoveCount m
        ON m.Discs = h.Hanoi_Disc_Count + h.Hanoi_Disc_Offset
        AND m.Pegs = h.Hanoi_Disc_Count
    WHERE NOT EXISTS
    -- Check for conflicting on target peg smaller than the one being moved.
    (SELECT 0
    FROM HanoiNpeg.view_Problem_Detail d
        WHERE (d.Disc_Peg IN (c.NumericValue, h.Disc_Peg))
            AND d.Disc_Size < h.Disc_Size
            AND d.ProblemId = @ProblemId)
            AND (Disc_Size <> Disc_Size_LastValue OR System_StepNumber = 0)
            AND c.NumericValue <> h.Disc_Peg
            AND c.NumericValue BETWEEN 1 and h.Hanoi_Disc_Count
            AND System_StepNumber < m.Moves
    -- For N disc, an offset defines the delta between the # of discs and pegs
    -- Pegs is set to hanoi disc count and actual disc count is disc count + offset

--
```

Figure 5-3: SQL Function for Next Move States for K-Peg

As the simulation finds solution paths, the framework captures the sequences associated to the solution (Table X). A unique feature of adding more pegs is that multiple optimal solutions arise, which is not the case for the 3-peg version. This requires UPRF to search across multiple solution instances in order to find a solution pattern and then cross-search within the k-peg solution space to correlate sequences for the variations. This represents a similar scenario but with multiple instantiations as the 3-peg variation. Table 5-1 provides sample move sequences and Table 5-2 provides example solution sequences that map closely to the 3-peg and from 4-peg to 5-peg as well as providing two diverse solution sequences.

Table 5-1: Sample Solution Sequences for K-Peg (peg count = disc count)

Hanoi_Disc_Count	Step	Disc	Peg
3	1	1	3
3	2	2	2
3	3	1	2
3	4	3	3
3	5	1	1
3	6	2	3
3	7	1	3
4	1	1	4
4	2	2	2
4	3	3	3
4	4	1	2
4	5	4	4
4	6	3	4
4	7	1	1
4	8	2	4
4	9	1	4
5	1	1	5
5	2	2	3
5	3	3	4
5	4	1	3
5	5	4	2
5	6	5	5
5	7	4	5
5	8	3	5
5	9	1	1
5	10	2	5
5	11	1	5

Table 5-2: Sample Disc Sequences for K-Peg Hanoi (disc count = peg count)

Discs/Pegs	Disc	Sequence	Value
3	1	----------1-1-1-1	85
3	2	----------1---1-	34
3	3	----------1---	8
4	1	----------1-1--1--1	329
4	2	----------1-----1-	130
4	3	----------1--1--	36
4	4	----------1----	16
5	1	----------1-1----1--1	1289
5	2	----------1-------1-	514
5	3	----------1----1--	132
5	4	----------1-1----	80
5	5	----------1-----	32
6	1	----------1-1-----1---1	5137
6	2	----------1---------1-	2050
6	3	----------1------1--	516
6	4	----------1----1---	264
6	5	----------1-1-----	160
6	6	----------1------	64

Table 5-3: Sample Peg Sequences for K-Peg Hanoi (disc-count = peg-count)

Discs/Pegs	Peg	Sequence	Value
3	1	----------1----	16
3	2	----------11-	6
3	3	----------11-1--1	105
4	1	----------1------	64
4	2	----------1-1-	10
4	3	----------1--	4
4	4	----------11-11---1	433

5	1	----------1--------	256
5	2	----------1----	16
5	3	----------1-1-	10
5	4	----------1--	4
5	5	----------11-111----1	1761
3	1	----------1----	16
		Alternate peg sequences	
3	1	----------1----	16
3	2	----------11-	6
3	3	----------11-1--1	105
4	1	----------1------	64
4	2	----------1--	4
4	3	----------1-1-	10
4	4	----------11-11---1	433
5	1	----------1--------	256
5	2	----------1----	16
5	3	----------1--	4
5	4	----------1-1-	10
5	5	----------11-111----1	1761
6	1	----------1----------	1024
6	2	----------1-----	32
6	3	----------1---	8
6	4	----------1--	4
6	5	----------1--1-	18
6	6	----------11-1111-----1	7105

Basic inspection shows that there is a definitive pattern for at least some cases of the 4-peg and 5-peg back to the 3-peg. Table 5-2 highlights the similarities. An area of further verification and research for UPRF would be to apply the same technique used in Chapter 4 for identifying the operator transform sequencing to progress from 3-peg to 4-peg, from 4-peg to 5-peg, and then from 5-peg to 6-peg. This provides data needed to generate the sequence transformation involved in generating the 4-peg prediction from the 3-peg prediction. The outcome should be a successful

prediction of a 7-peg solution sequence. With sufficient transformation learning, a general solution should arise just as with the 3-peg scenario.

The k-peg scenario tables include the binary values from the sequences. The work in Chapter 3 did not include this, but the principle is applicable to all sequencing solution scenarios. Their use here calls out a potential area of research for utilizing learning operators to correlate the values associated to the binary sequences rather than the sequences. There is an algebraic relationship between the numbers as Table 5-4 illustrates that the total number of state changes add up to the total number of state changes mandated by the goal. A learning operator could deduce the sequence of operations for a missing peg or missing disc once the other peg or disc sequences were determined by subtracting the values of the determined sequences from the value of the total sequences. For example Table 5-4 shows that the total value of all sequences for the 4-peg scenario is $2^9 - 1$ (511). If the sequence values for pegs 1, 2, and 4 were determined for the first sequence example (64, 10, 433), then one can derive the third sequence by subtracting (501 (64 + 10 + 433) from the total required for the overall sequence (511) and arrive to 10 as the sequence value for Peg 3 which expands to the binary sequence 1010.

Table 5-4: Total Sequence Values for Pegs or Discs

Discs/Pegs	Sequence	Value
3	----------1111111	127
4	----------111111111	511
5	----------11111111111	2047

The patterns in Table 5-2 clearly show a pattern relationship as the number of pegs increase for the same number of discs. In this case, the pattern for the discs is easy to derive, as it is merely the widening of the bit strings in a symmetrical fashion. A pattern that predicts the cascading of

the pegs also emerges in Table 5-3. The test case was for using an equal number of discs as pegs. Undoubtedly, similar patterns emerge from adding additional discs for the same number of pegs.

This example provides support for the concept of extending the learning capability of UPRF by expanding the variability of a problem. The same sequencing problem arises from the k-peg as the 3-peg scenario. A similar learning scenario for sequence inspection with higher-level transformation operator sequencing emerges as that associated with the automated general solution discovery of the 3-peg scenario.

5.1.2 Multi-agent Scenario – Two Player Tic-Tac-Toe

The bi-direction allows the framework to incorporate other problems as entities within larger problems to support virtually any type of problem scenarios. The architecture of UPRF finds solutions through seeking to meet a goal. In Hanoi, the goal was deterministic and absolute in terms of either failure or success. However, there is nothing in the structure preventing UPRF from seeking solutions and transforming such solutions into higher order problems in the same way as pursuing the Hanoi transformational sequence. This section models Tic-Tac-Toe as an example of a multi-agent scenario.

For the purposes of this chapter, a multi-agent scenario refers to a problem that is either collaborative, competitive, or a combination of the two. A collaborative scenario involves multiple agents working toward a single goal. A competitive scenario involves one or more agents competing against each other to reach a goal and includes the scenario of multiple agents working together against another team of multiple agents. The polymorphic database design of UPRF provides the support needed for all of these scenarios.

Figure 5-1 illustrates a sample schema for Tic-Tac-Toe. In the Tic-Tac-Toe scenario, the goal is to find the optimal solution path from both player's perspective. Once the transformational sequence processing done, the goal is for the simulation to perform the optimal moves for each player from learning the brute-force simulations. In Tic-Tac-Toe, the significant patterns are all within only three significant squares for the start – the center square, a corner square, and a mid-point square along a row. Therefore, a simulation process that explores paths for these three different squares should yield the transformation sequence to automatically find the optimal solution path for the other six starting squares and ultimately identify the correct responses to avoid the failure state and maximize achievement of the goal state. The schema in Figure 5-1 provides enough information to instantiate all of the possible simulations including redundant ones by varying the row range from one to three as well as the column range. In addition, the concept of a player is added which varies from one to two. This allows the problem to vary by player creating separate simulations from the perspective of the different players with the goal being relative to the player number of the simulation. Figure 5-2 identifies the constraints of the move, goal, and failure states so that the simulation can proceed similar to Hanoi.

```
<?xml version="1.0" encoding="UTF-8"?>
- <Upr xmlns:xsi="ProblemDef-v8.xsd" Version="11.0">
        <!-- This problem illustrate how UPR can work for multi-player -->
    - <Problem PlayQuery="Query-Next-Disc-Peg" FailQuery="Check-if-Lost" GoalQuery="Check-If-
      Won" Id="Tic-Tac-Toe">
        - <Attribute Id="Step-Count">
            <Expression>Get-Step-Count</Expression>
          </Attribute>
        - <Attribute Id="Square-Range">
            <Range VaryUsing="ADD" VaryBy="1" VaryTo="3" VaryFrom="1"/>
          </Attribute>
        - <Attribute Id="Player-Number">
            <Range VaryUsing="ADD" VaryBy="1" VaryTo="2" VaryFrom="1"/>
          </Attribute>
        - <DataEntity Id="Player-Move" DeriveFrom="Player-Number">
            <Attribute Id="Row" Domain="Square-Range"/>
            <Attribute Id="Col" Domain="Square-Range"/>
          </DataEntity>
            <!-- All attributes, problems, and entities are automatically generated as expressions and
            available for use as expressions in the database -->
          <Expression Id="Row-Total" Attribute="Row" Operator="COUNT"/>
          <Expression Id="Col-Total" Attribute="Row" Operator="COUNT"/>
          <Expression Id="Move-Count" Attribute="Player-Move" Operator="COUNT"/>
          <Expression Id="Step-Count" Attribute="Step-Count" Operator="SELECT"/>
          <Expression Id="Next-Player" Attribute="Player-Move" Operator="SELECT_NONUSED"
            NullExpression="1" Step="0"/>
          <Expression Id="Last-Player" Attribute="Player-Move" Operator="SELECT_USED"
            NullExpression="1" Step="0"/>
          <Expression Id="Nonused-Rows" Attribute="Row" Operator="SELECT_NONUSED"/>
          <Expression Id="Nonused-Cols" Attribute="Col" Operator="SELECT_NONUSED"/>
```

Figure 5-4: Sample Tic-Tac-Toe Problem Schema – Initial Relations

```
    <Expression Id="Nonused-Cols" Attribute="Col" Operator="SELECT_NONUSED"/>
  - <Query Id="Play-Game" ColumnName="Start-Player" BaseExpression="1">
    - <Lookup ColumnName="Player-Number" OutputAttribute="Player-Move"
      Expression="Next-Player">
            <!-- Determine the current player-->
      </Lookup>
      <Lookup ColumnName="Next-Row" OutputAttribute="Row" Expression="Nonused-
        Rows"/>
    - <Lookup ColumnName="Next-Row" OutputAttribute="Col" Expression="Nonused-Cols">
        <Criteria Attribute="Row" CompareOperator="EQUAL" QueryColumn="Next-Row"/>
          <!-- Lookup the columns not yet used for the non-used row-->
      </Lookup>
    </Query>
      <!-- Set the goal of the game for each player to win to exhaust all successful paths This will
      provide the data for a learning game to find the optimal moves to maximize chance of winning
      for both sides -->
  - <Query Id="Check-If-Won" ColumnName="Player-Number" BaseExpression="Player-
    Number" OutputAttribute="2">
          <!-- Return status of 2 means flag as successful solution but continue to simulate
          variations until all paths exhausted-->
    - <Lookup ColumnName="Row-Total" Expression="Row-Total">
        <Criteria Attribute="Player-Move" CompareOperator="EQUAL" QueryColumn="Player-
          Number"/>
      </Lookup>
    - <Lookup ColumnName="Row-Total" Expression="Col-Total">
        <Criteria Attribute="Player-Move" CompareOperator="EQUAL" QueryColumn="Player-
          Number"/>
      </Lookup>
    - <Lookup ColumnName="Row" Expression="Row">
        <Criteria Attribute="Player-Move" CompareOperator="EQUAL" QueryColumn="Player-
          Number"/>
      </Lookup>
      <Lookup ColumnName="Col" Expression="Col"/>
    - <Lookup ColumnName="Diag-Total" Expression="Move-Count">
        <Criteria Attribute="Col" CompareOperator="EQUAL" QueryColumn="Row"/>
      </Lookup>
      <Filter Id="Check-3-In-Same-Col" Operator="EQUAL" Column2="3" Column1="Col-
        Total"/>
      <Filter Id="Check-3-In-Same-Row" Operator="EQUAL" Column2="3" Column1="Row-
        Total" OrFilter="Check-3-In-Same-Col"/>
      <Filter Id="Check-3-In-Same-Diag" Operator="EQUAL" Column2="3" Column1="Diag-
        Total" OrFilter="Check-3-In-Same-Col"/>
    </Query>
  </Problem>
</Upr>
```

Figure 5-5: Tic-Tac-Toe Query Rules

In the Tower of Hanoi, there was an exhaustive review of multiple simulations including the state sequences. All simulations within UPRF follow the same model of depth-first solution space exploration, so it is not necessary to examine Tic-Tac-Toe exhaustively, but it is useful to explore

a sample showing the interaction from both the starting player and responding player perspective. This example represents an evolution of the schema to support the constructs of Tic-Tac-Toe more easily. However, this evolution is not a reflection that the Hanoi schema is lacking, but rather an improvement more relevant to helping for this scenario. The actual data structure utilized to store the schema is identical in each version. This section explores the progression in terms of the following phases from both player perspectives:

1. Instantiation Phase: Generates the initial scenario for placement of the first square for the first player for all the possible first set of moves.
2. Play Phase: Generates the response moves from perspective of both players as separate simulations. Sequences for the relational states are captured in this phase.
3. Transformation Phase: Maps the sequences of operations to the higher order transformation problem.

5.1.2.1 Instantiation Phase

The instantiation phase creates the following instances by nature of the expressions embedded in the problem definition using the attribute overflow concept explained earlier in the dissertation. The attribute overflow principle means that whenever more a query or expression generates more than one row of data, generation of a new simulation instance arises that represents that unique path. Based on this, the output is a combination of the following values:

- Players: 1 or 2 (Generated by the Range construct for the Player Number attribute)
- Player Move
    - Player (Derived from Player Number)
    - Row: 1 through 3 (Derived from Square-Range rule)

- Column: 1 through 3 (Derived from Square-Range rule)

5.1.2.2 Play Phase

In the play phase, the simulation applies the goal context based on the player role associated to the simulator against the grid of square representing the plays made including the coordinates as well as the player associated to the move. The Player-Move entity thus includes not only the coordinates but also the player that made the particular move. The rule accomplishes this through the Play-Game query, which looks for a non-used square and non-used column to assign the next move. The outputs of Play-Game are thus:

- Player Number making the move based on a lookup that forces the player number to alternate on each move.
- Square selected
- Column selected

The framework thus generates overflow situation necessary for branching multiple instances for every possible move for each player. This generates the relational state sequences that represent the relationship of each variable values relative to the sequence at which the value changes.

5.1.2.3 Transformation Phase

The transformation phase occurs after achieving solutions to two distinct instances. This phase regenerates the problem of deriving an instance solution from another instance solution. The problem is modeled executing transform operators to try to generate a sequence of operators that successfully transform the first instance operations to create the operations used to solve the second instance.

In the case of Hanoi, the solving of the solution path for different instances was simplified since each solution path ultimately derives from the same recursive algorithmic solution with only a slight variation based on whether the count of discs for the instance was even or odd. With Tic-Tac-Toe, some paths are more successful than others are. For example placing a square in the center peg ensures at least a tie-game ("Cat's game") for the first player and results in several scenarios where the first player is victorious. However, placing a square in a diagonal square, while still effective to ensure at least a tie, does not generate as many victorious paths and the sequence to success is different.

The outcome of multiple-solution path instances is that the transformation operators should eventually find a sequence that converges on common variables in the same way as Hanoi. Ultimately, with Hanoi, the transformations become more and more abstract such that by the third transformation, a very simple set of operators are able to generate the lower level solution to posit the higher order problem – solving it generically. The same approach works for Tic-Tac-Toe with the caveat that there is "noise" which serves to invalidate some instance as not related to other instances. For example, Player 1 responses to Player 2 placing a mark in a corner square on the first move indicate a different solution path than if Player 2 places a mark in a middle row or column. However, if the response of Player 2 is simply a transformation of another response across a different axis, the solution patterns should be convergent. For example, if Player 2 response to Player 1's first move in the center with an adjacent square rather than a square diagonal, the solution paths are deterministic relevant to symmetry. Table 5-5 illustrates the concept of related instances with solution paths whose variance is purely a function of symmetry as opposed to other instances whose solution path is not attributable to symmetry.

Table 5-5: Solution Symmetries in Tic-Tac-Toe

O	X				X	O
O						O
	X				X	

In the above chart, the olive-shaded instances have symmetric optimal solution paths separate from the solution patterns for the beige-shaded instances.

To examine all the potential solution paths exhaustively using combinations of Player 1 and Player 2 would take hundreds of lines of relational state sequence captures. A single base pattern with different symmetries illustrates the relational state sequence chapter and how the transformation problems evolve to converge on a solution transformation sequence that solves multiple instances across symmetries. Based on this, the framework targets four instances initiated by Player 1 moving to the center but with asymmetrical responses from Player 2. This is a subset of the possible paths, but it illustrates the learning transformation process. By the end of the simulation, UPRF is able to generate the solution to the fourth sequence from transformation without the use of simulation. Table 5-6 shows the square labeling convention. These instances are:

- P1: R2,C2; P2: R2,C1
- P1: R2,C2; P2: R2,C3
- P1: R2,C2; P2: R1,C2

- P1: R2,C2; P2: R3,C2

Table 5-6: Tic-Tac-Toe Square Numbering

R1, C1	R1, C2	R1, C3
R2, C1	R2, C2	R2, C3
R3, C1	R3, C2	R3, C3

Figure 5-6 illustrates the transformation process relative to the Player 2 response. This diagram is very similar to the approach from Hanoi. The difference is that only two levels of transformations are necessary to solve the fourth instance given the constraints outlined for a similar solution path with different symmetries.

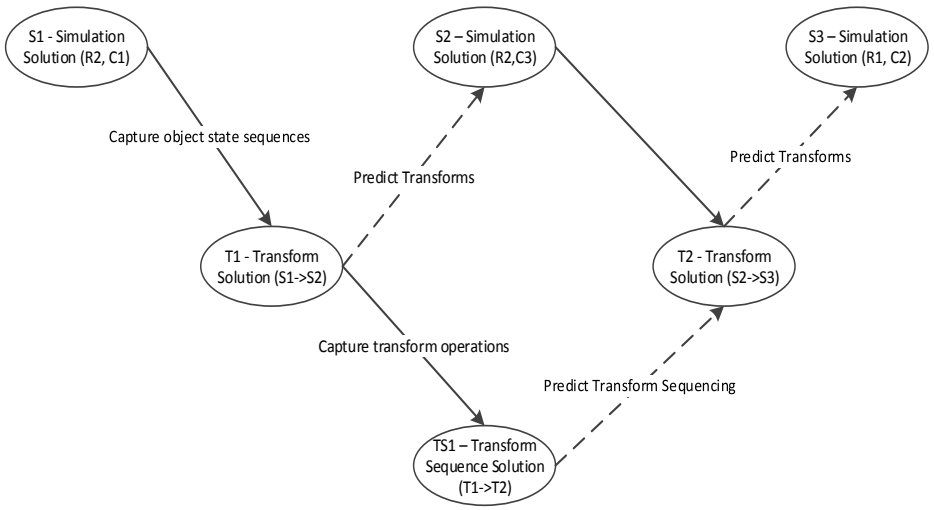


Figure 5-6: Tic-Tac-Toe Simulation Transform Map

The generic solution for the symmetrical sequence used to achieve victory in S1 derives from the transformations as follows:

- S1, S2 -> T1
- S2, S3 -> T2
- T1, T2 -> TS1

TS1 will contain the generic operators to generate T2 that transforms S3 to S4 without the need for simulation. The goal is that by solving three instances through simulation, the framework learns the fourth instance transformation generating the solution sequence without simulation. Figure 5-7 shows the generation of simulation four from the learned sequence from the third transform generated by the generic transform sequence solution. The model will increase in depth to support more advanced transformations including how to determine the method for determining the correct response to different variations as instances are added with non-symmetric responses. This was examined in detail in Hanoi and follows for Tic-Tac-Toe and all other problem scenarios.

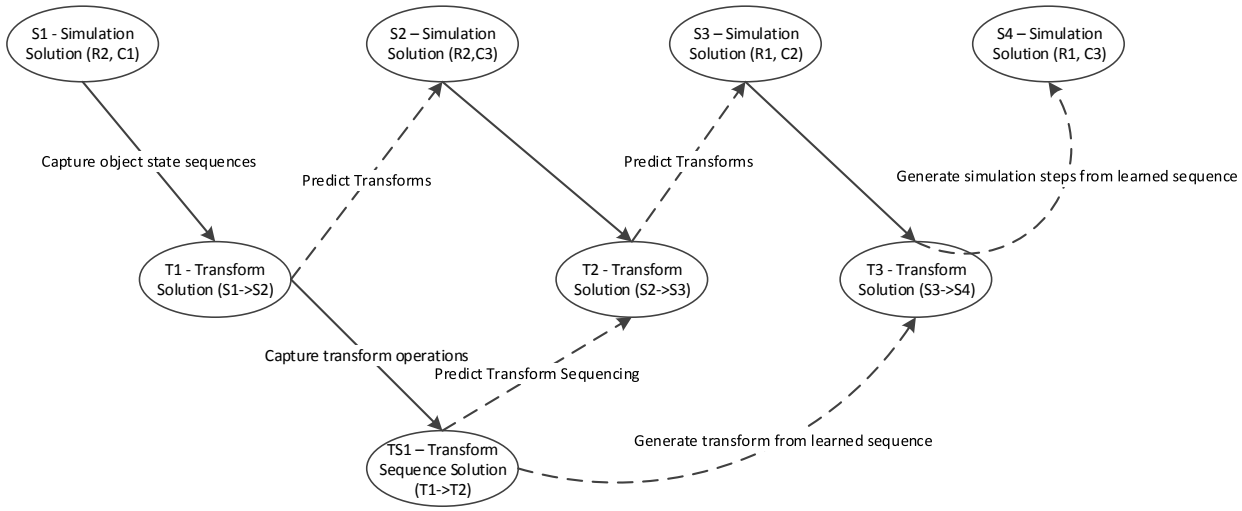


Figure 5-7: Tic-Tac-Toe Simulation Map with Sequence Generation

The base scenario is the forced victory that comes from Player 2 moving to an adjacent square rather than the corner. Table 5-7 shows the move sequence pattern for the first three scenarios that provide the information ultimately needed to generate the fourth scenario solution.

Table 5-7 Tic-Tac-Toe Base Use Case for Simulation 1 (S1)

R1, C1 3	R1, C2 5	R1, C3 6
R2, C1 2	R2, C2 1	R2, C3
R3, C1	R3, C2 7	R3, C3 4

Tables 5-8 and 5-9 depict the transformations as simple rotations of the first simulation:

Table 5-8 Tic-Tac-Toe Base Use Case for Simulation 2 (S2)

R1, C1	R1, C2 2	R1, C3 3
R2, C1 7	R2, C2 1	R2, C3 5
R3, C1 4	R3, C2	R3, C3 6

Table 5-9 Tic-Tac-Toe Base Use Case for Simulation 3 (S3)

R1, C1 4	R1, C2 7	R1, C3

R2, C1	R2, C2 1	R2, C3 2
R3, C1 6	R3, C2 5	R3, C3 3

Using Table 5-10 and applying symmetric transformations yields relational state sequences for the first three scenarios that mirror one another. The table shows that each pattern repeats in the other instances by varying the square and column that reuses the sequence. All that is necessary to generate a transform sequence is to identify the variation that drives the transformation. The following transforms occur for S1 - > S2:

- Row 1 -> Column 3
- Row 2 -> Column 2
- Row 3 -> Column 1
- Column 1-> Row 1
- Column 2 -> Row 2
- Column 3 -> Row 3

Thus, an operation sequence that transforms Rows to Columns and adjusts the column numbers inverse to the row numbers generates the solution sequence for S2.

For S2 -> S3:

- Row 1 -> Column 3
- Row 2 -> Column 2
- Row 3 -> Column 1
- Column 1-> Row 1
- Column 2 -> Row 2
- Column 3 -> Row 3

The same approach works for S2 to S3. Therefore, the same sequence of transform operators can predict S4 and the solution is generic for the symmetry. The simulator can then apply this learned knowledge to generate higher order transforms for other symmetries. Ultimately, the symmetries feed up such that the framework generates a solution that defines the operations required for each sequence of moves to transform to the optimal solution.

Table 5-10: Sample Solution Sequences for Tic-Tac-Toe

Attribute	Value	State Change Sequence
		S1
Player	P1	1-1-1-1
Player	P2	-1-1-1-
Row	1	--1-11-
Row	2	11-----
Row	3	---1--1
Column	1	-11----
Column	2	1---1-1
Column	3	---1-1-
		S2
Player	P1	1-1-1-1
Player	P2	-1-1-1-
Row	1	-11-----
Row	2	1---1-1
Row	3	---1-1-
Column	1	---1--1
Column	2	11-----
Column	3	--1-11-
		S3
Player	P1	1-1-1-1
Player	P2	-1-1-1
Row	1	---1--1
Row	2	11-----
Row	3	--1-11-
Column	1	---1-1-
Column	2	1---1-1
Column	3	-11----

From the above, S4 with an initial move of R1, C3 by Player 2 follows directly from sequence transformation if the playing pattern is the same in regard to symmetry.

5.1.3 Advanced Multiple-Agent Scenarios

The Tic-Tac-Toe example illustrated the exploration of a solution path that involves multiple agents being the players. Each player is simply another variable with an associated state sequence mapped against a minimal energy efficiency goal to obtain success. In this way, an exhaustive simulation with upper level transformations will yield the ideal move patterns for both players.

However many multi-agent scenarios are far more complex and not deterministic. Can the same concepts be applied to games or problem scenarios that include more than one antagonist? What is the impact on the usability of the framework if multiple agents collaborating against another team is introduced? For these questions, this section considers two simple games that involve more than two agents: Chinese checkers and Canasta.

Chinese checkers may have more than one player whose goal is to change the state of their objects to be within the opposing player's square prior to other players. However, the pursuit of the goal is more complicated because of the interaction with other players. Interaction with other player moves may both benefit and hurt the changes of a player meeting the goal state. In the Tic-Tac-Toe scenario, players were modeled as variables. This is one approach to multi-agent, but there is another approach that works within UPRF that allows the perspective of each player to be the actual problem. This allows the UPRF to model the optimization problem for each player as a separate problem from the perspective of the particular player achieving the goal. The polymorphic database design of UPRF provides the role-perspective ability. In UPRF, every object may be polymorphic. This allows the framework to represent a problem in one scenario as an entity inside of another problem. For example, the framework can represent the interactions occurring on one

simulation involving a separate player as an entity state sequence within another problem. This allows simulation to occur in parallel on behalf of each player.

Figure 5-8: Chinese checkers

The team version of the card game Canasta is an example of a multi-player scenario where two members of the same team collaborate against another team. This use case benefits from UPRF's ability to instantiate a problem in terms of multiple players as entities on the same team pursuing a goal in parallel to solving the same problem for a different team that interacts as an opposing entity. In this approach the schematization of entities include both players on the same team to monitor state sequences that contribute towards the goal while defining the paths derived from the actions of another simulation mirroring the opponents' actions as a separate entity attribute sequence. The framework can use the same technique for any collaborative/competitive scenario.

5.1.4 Non-deterministic and highly complex type problems including NP-complete

This section examines scenarios where no deterministic path may be found or the most optimal solution remains undiscovered. This section also reviews probabilistic problems that may not have any optimal solution. This section examines the following problem scenarios:

- **Zero-subset sum problem** – This is an NP-complete problem that is easy to model in UPRF. This problem is chosen because all NP-complete problems are reducible to other manifestations. The optimal solution to zero-subset thus can benefit not only zero-subset but also other NP-complete problems including the TSP.
- **Stock market trade optimization** – This probabilistic scenario investigates how UPRF can schematize problems that closely model real-world non-deterministic scenarios. This problem endeavors not only to identify optimal parameter and parameter value selection to maximize profitability for a given window of time, but also to identify an autonomous learning process whose execution resides above the profit-making simulation process over time periods. The problem does this in order to identify the optimal methods to discover parameters and parameter value variation methods to obtain maximum benefit over the lifetime of a trading strategy.

5.1.4.1 Zero-Subset Sum Problem

In the zero-subset sum problem, the goal is to find a subset of integers within a set whose sum is zero. Modelling the zero subset sum problem as a binary problem allows for representation in UPRF [122]. The number of steps required to find the integers is exponential even though the solution can be determined in polynomial time. The inability to solve a problem in polynomial time even though validation of the solution is possible in polynomial time is a distinguishing attribute of NP-complete problems [123]. In this section, the framework represents the zero-subset problem schematically so that simulation can generate possible instances of the problem within a range and then attempt to determine through simulation the optimal adding sequence to add the numbers for all possible number sets within a test range. The learning transformation problem is

the same as in prior scenarios. As the framework solves each subsequent instance, it generates a transformation problem to determine the transform operators that can generate the sequence of solving steps from one instance using another instance. The framework transforms each successful transformation solution into a higher order transformation problem to generate the transformation sequence for one instance from another instance.

The modelling of zero-subset here does not intend to resolve the issue of P=NP, but shows how the process contributes toward optimizations for the problem. For the purposes of illustrating the approach, the transformation process is limited and does not involve many instances. A potential contribution of an exhaustive problem solving exercise in this area is a proof relevant to P=NP. If it can be shown that, with sufficient instances of problem configurations, the problem solver converges to all of the known optimizations and the complexity remains non-polynomial, this is a compelling argument for P!=NP. On the other hand if some point of equilibrium is reached based on discovery and application of optimizations into the transformation sequences such that the complexity becomes less than polynomial, this establishes P=NP. It is not the purpose of this exercise to answer the question or to provide a proof, but to outline how UPRF can pursue an NP-complete problem such as zero-subset sum to seek the optimal solution.

As in prior scenarios, the first step is to schematize the problem. Figure 5-9 illustrates the schematization using a version of the problem schema that works well with modelling this problem but still maps to the generic database structure proposed earlier. Figure 5-10 depicts the schema used. Not all features of the schema are used. The “Play” query is not needed because the initial setup query already generate all of the solved instances so these are immediately ready for transformation to a higher order correlation problem. Due to the polymorphic nature of a problem

persisted as a problem entity within itself, the framework can model even a problem that has no sub-entity steps for transformation in terms of its setup query outputs. The problem definition including its attributes are stored polymorphically in the schema as an entity which allows visibility in the engine to any attributes related to the problem in the same method as sub-problem entities. Another feature of this schema is the concept of a junction that contains criteria applying to expressions separately. This functionality supports nested conjunction such as having a set of AND operators within a set of OR operators. This simplifies the use of Boolean expressions with conjunctions to support any type of extraction from the query and underlying expressions.

```
- <Upr Version="13.0">
  - <Problem Id="Zero-Subset">
        <!--Problem Control Attributes -->
      <Attribute Id="NumberCount"/>
        <!-- Count of numbers for the subset problem-->
      <Attribute Id="AddingSequence"/>
        <!--Unique Sequence to use for adding values together-->
    - <DataEntity Id="IntegerIndex">
        <Attribute Id="IntegerValue"/>
      </DataEntity>
    - <Query Role="Setup">
      - <Expression Id="GenerateNumberCount" Operator="GENERATE_RANGE">
          <Argument Value="2" Parameter="MinValue"/>
          <Argument Value="4" Parameter="MaxValue"/>
            <!-- To limit the initial test-->
        </Expression>
      - <Expression Id="GenerateNumbers" Operator="GENERATE_RANGE">
          <Argument Value="GenerateNumberCount" Parameter="ValueCount"/>
          <Argument Value="-2" Parameter="MinValue"/>
          <Argument Value="2" Parameter="MaxValue"/>
          <Argument Value="RANDOM" Parameter="Method"/>
          <Argument Value="VALUE" Parameter="OrderBy"/>
          <Argument Value="ALL_POSSIBLE" Parameter="Scope"/>
          <Argument Value="1" Parameter="Unique"/>
            <!-- Generate all possible combinations of non-zero numbers-->
        </Expression>
      - <Expression Id="GenerateAddingSequence" Operator="GENERATE_SEQUENCE">
          <Argument Value="GenerateNumberCount" Parameter="ValueCount"/>
          <Argument Value="0" Parameter="MinValue"/>
          <Argument Value="1" Parameter="MaxValue"/>
          <Argument Value="RANDOM" Parameter="Method"/>
          <Argument Value="KEY" Parameter="OrderBy"/>
          <Argument Value="ALL_POSSIBLE" Parameter="Scope"/>
          <Argument Value="1" Parameter="Unique"/>
            <!-- Generate all possible combinations of sequence for adding the numbers -->
        </Expression>
        <Output Target="NumberCount" Source="GenerateNumberCount.Value"/>
        <Output Target="IntegerIndex" Source="GenerateNumber.KEY"/>
        <Output Target="Number.IntegerValue" Source="GenerateNumber.VALUE"/>
        <Output Target="Zero-SubSet" Source="GenerateAddingSequence.KEY"/>
        <Output Target="Zero-SubSet.AddingSequence" Source="GenerateAddingSequence.VALUE"/>
      </Query>
        <!-- No play query is needed, because instances generated in setup represent all possible solution paths for all
        possible sets of numbers within the range -->
    - <Query Role="Goal">
      - <Expression Id="NonZeroNumbers" Operator="FILTER" Source="IntegerIndex">
          <Argument Value="NOT_EQUAL" Parameter="FilterOperator"/>
          <Argument Value="IntegerValue" Parameter="FilterAttribute"/>
          <Argument Value="IntegerValue" Parameter="Filter"/>
        </Expression>
      - <Expression Id="CountFlaggedSequenceValues" Operator="COUNT_FLAGGED_VALUES_IN_SEQUENCE"
        Source="Zero-Subset">
          <Argument Value="AddingSequence" Parameter="Sequence"/>
          <Argument Value="IntegerIndex" Parameter="Key"/>
          <Argument Value="IntegerValue" Parameter="Value"/>
            <!-- Counts the Flagged Values -->
        </Expression>
      - <Filter>
        - <Junction>
            <Criteria Expression2="0" FilterOperator="EQUAL" Expression1="CountFlaggedSequenceValues"/>
          </Junction>
        </Filter>
        <Output Target="SOLUTION_STATE" Source="2"/>
          <!-- 2 indicates problem solved which will only exist if filter passes-->
      </Query>
    </Problem>
  </Upr>
```

Figure 5-9: Zero-Subset Sum Problem Schema

Figure 5-10: Updated UPRF Schema (v13)

The baseline problem definition thus generates not only all possible test values for a domain of numbers but all of the sequences that might be used to solve the problem. The problem could have been defined in the playing query to integrate known optimizations such as that from Horowitz and Sahni [124] which reduces the time to $O(2^{N/2})$ rather than using brute force. However, schematizing the problem in the current manner provides an opportunity to validate UPRF's optimization process. The framework ultimately converges to the most optimal method to solve the problem as successful simulation sequences from different instances promote to transformational problems.

Applying the concepts from the prior solving exercise shows that UPRF will converge to the optimal transformational sequence. The progression for achieving this is:

1. Solution instances will all have at least one negative number and one positive number in order to generate the zero subset. The framework identifies this by correlation of the engine to the factors relevant to the solution instances. This is a feature not exposed by prior exercise, but is clearly easy to implement into the framework by modelling a problem whose goal is to eliminate sequences that do not generate a solution and correlate the data values to the failed instances. The framework can then assert this optimization back into the original problem as a failure query to speed up the simulation process.
2. Positing a higher order problem against the base problem applies operators to transform successful instances to one another. A set of transform operators provides the domain for which to register selection of an algorithm given the inputs. There are

definitive correlations associated with optimizations involving the order of numbers tested within a range.

3. The higher order problem identifies a pattern for testing the sequences from the sequence of numbers flagged for inclusion in the subset problem that correlates across instances. This becomes a third order higher problem and once this is resolved, the framework will establish the optimal way to sequence the testing of the numbers for inclusion in the subset calculation. Utilizing different functions for selecting the sequence provide the candidate transformational operators.

UPRF can utilize a transformation problem that correlates solved instances incrementally where different functions define the sequencing of the numbers for testing. UPRF is limited to transform operations that provided to the framework. This is an efficiency issue and not a functionality issue. After enough iterations, the framework will establish a variable relationship that replicates the partitioning function through a sequence of more primitive operations so long as the primitive operations are sufficient to construct the higher-level function. This assertion comes from the postulate that UPRF converges to complete correlation relevant to the transform operators available.

Examination of the outputs of the problem instantiation in Table 5-11 shows the correlation to the sequencing utilized to reach the goal. The combination of the number set and the solution sequence generates a unique sequence. The framework can then use this sequence as a base instance for transform operators to recognize the operators that converts one sequence to another. Table 5-1 shows the state sequences for two different instances. The two instances reveal the same optimal solution sequence for different numbers in the solution. This provides information to the

learning algorithm in the transformation problem to identify the correlating factors between the two simulations. In this case, the major correlating factor is that the numbers at both extremes of the second sequence are respectively decremented and incremented by the same amount. The information learned from this allows UPRF to solve a new problem instance that has this variation instantly without the need for simulation.

Table 5-11: Sample Zero-subset Solutions

Numbers	Count	- Sum	+ Sum	Solve Sequence	Observation
2,-1,1,2	4	3	3	1,2,3,4	A: Positive Sum = Negative Sum -> Add all numbers
				1,4	B: All numbers symmetrical domain -> Choose equal offset from mid points C: Choose opposites for immediate match
				2,3	
-2,-1,1,2		2	2	1,2,3,4	A
				1,4	B
	4			2,3	B
-2,-1,1,3		-3	4	1,2,4	
	4			2,3	C
-2,-1,2,3		-3	5	1,2,4	
	4			1,3	C
-2,1,2,3	4	-2	6	1,3	C
-1,1,2,3	4	-1	6	1,2	
-2,1,2	3	-2	3	1,3	C
-2,-1,1	3	-3	1	2,3	C
-3,2,3	3	-3	5	1,3	
-1,1,2	3	-1	3	1,2	C
-2,2,3	3	-2	5	1,2	C

Table 5-12: Identical Solve Sequence for Different Subset Problems

Attribute	Value	State Change Sequence
Set: -3, -1, 2, 4 Representation		
Solve Sequence	1,2,4	
Index	1	100

Index	2	010
Index	3	000
Index	4	001
Value	-3	100
Value	-1	010
Value	+2	000
Value	+4	001
Set: -4, -1, 3, 5 Representation		
Solve Sequence	1,2,4	
Index	1	100
Index	2	010
Index	3	000
Index	4	001
Value	-4	100
Value	-2	010
Value	+3	000
Value	+5	001

5.1.4.2 Stock Market Trading Optimization

The stock market profitability scenario represents a probabilistic use case that is not of definitive determinism. Some research cites the efficient market theory that over a long enough period of time, there is no actual algorithmic approach that can fare better than simple guessing [125]. The goal in modelling stock-market trading as a problem to UPRF is not to determine if that is the case, although this could be a special type of problem for the system to solve. The goal is to determine if UPRF provides a framework for optimizing performance of a stock trading portfolio through the optimization process.

Modelling the stock market trading scenario represents a use case where data already defined in a format conducive to analyzing and reporting on the stock market. For this scenario, UPRF executes in an external fashion to integrate existing data against the rules to seek for the simulation. Implementing UPRF for this scenario requires considerable software infrastructure development

that is not present at this time in the project. However, using a system that conceptually implements UPRF helps demonstrate the feasibility of the optimization process.

Modelling of the stock market trading scenario occurs at two levels. The first level is simply to test various parameters with variable values in a brute force method to generate different portfolios and test their performance. This represents the brute force simulation exercise of UPRF that gathers data so that the upper level problem transformation can occur. This scenario incorporates the concept of autonomous learning ("auto-learning") to demonstrate problem transformation of lower level instances. This is a variation from the sequencing approach in order to add another variable for applying the solution from the lower level instances to the higher order problem of identifying the most relevant profitability trend.

In the auto-learning approach, the system calculates profitability window based on parameter ranges for time to look back and number of days in the evaluation period. Based on this, a higher order problem implements this as the measurement mechanism to test for a trend change. At a yet higher level, a simulation occurs which applies various profitability windows to how to determine the profitability window. Figure 5-11 illustrates the progression that feed the results of the lower level simulations as inputs for higher-level simulations that utilize parameters for trending profitability. Figures 5-12 and 5-13 depict schemas to enable integration with the simulation process as well as for the auto-learn. Figure 5-14 shows an additional schema oriented particularly towards modelling a behavior. This system is still in development but provides an opportunity to integrate an analytic application with UPRF.

Initial test results of UPRF concepts with the stock trading application known as "CapGen" show promise. The CapGen database currently stores over 3 billion records downloading over 2

million quote rows daily for both intraday and end-of-day. In addition, the system generates several million technical indicator values in a key/value store. The system provides a generic external data collection interface that includes web-scraping functionality to access sentiment data including social media feeds and macro data (Figure 5-20). Collected indicators as well as strategies themselves persist into the system as index instruments along with trade-able equities allowing correlation to occur among not only assets but also indicators. This abundance of data provides an excellent foundation for UPRF to operate against in simulative and learning processes. Strategies are loaded into the system using a schema specific to CapGen (Figures 5-18 and 5-19) to represent the concept of triggering instruments with parameters as well as the trade execution instruments and parameters. This structure is transformable to the UPRF schema to represent queries for the overall problem solving of optimizing asset purchase and sales. Initial results are promising. Figure 5-22 shows the impact of optimizing an existing strategy based on auto-learning of cycle data to improve the results significantly higher than that of a buy-and-hold strategy on the instruments. In this case, the Auto-learn varied parameter values correlated to greed and fear by setting exit and stop limits that align with volatility changes in the market over the testing period running from 1992 through most of 2013. The approach shows how the higher-order learning approach of UPRF learns and adapts to avoid over-fitting data from previously learned intervals.

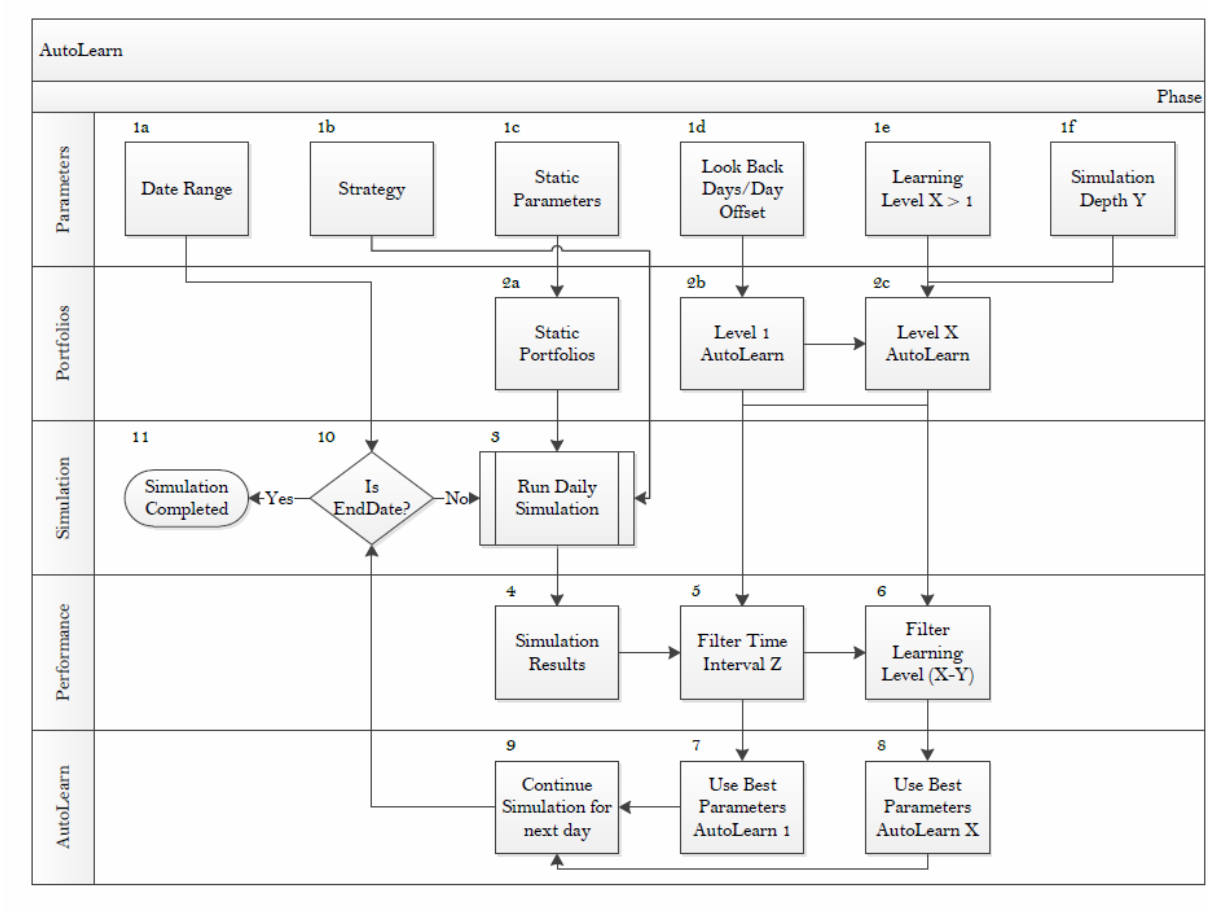


Figure 5-11: Incremental Trading Strategy Improvement

```
<?xml version="1.0" encoding="UTF-8"?>
<Upr xmlns:xsi="ProblemDef-v8.xsd" Version="11.0">
        <!-- This problem illustrate how UPR can try to find an optimal stock trading strategy
        Generates a portfolio that is associated to a triggeringequity, metric, metric ratio selection, and
        purchasing equity. This problem generates data which is then used by the auto-learn problem
        to automatically select the best strategy based on performance. The auto-learn problem applies
        different parameters for how to select the best strategy. This generates data for yet a higher-
        level problem that determines the best intervals to use for evaluating the strategies -->
    - <Problem DeriveStepFrom="Trading-Days" PlayQuery="Trade-Market" Id="Stock-Trader">
        <Attribute Id="Profitability-Ranking"> </Attribute>
      - <Attribute Id="Metric-List">
            <Expression Id="Metrics"/>
        </Attribute>
      - <Attribute Id="Metric-Ratio-List">
            <Range VaryUsing="ADD" VaryBy="Ratio-Variation" VaryTo="1.5"
                VaryFrom=".5"/>
        </Attribute>
      - <Attribute Id="Ratio-Variation">
            <Value>.1</Value>
        </Attribute>
      - <Attribute Id="Trading-Symbol-List">
            <Expression Id="Trading-Symbols"/>
        </Attribute>
      - <Attribute Id="Trigger-Index-List">
            <Expression Id="Trading-Indices"/>
        </Attribute>
      - <Attribute Id="Daily-Budget">
            <Value>5000</Value>
        </Attribute>
      - <Attribute Id="Total-Budget">
            <Value>50000</Value>
        </Attribute>
      - <Attribute Id="TradingOptimizerDb">
            <TextValue>server=folidc06;initial catalog=tpv5</TextValue>
                <!-- Points to data source row for connection string -->
        </Attribute>
      - <DataEntity Id="Portfolio-Index-Transaction" DeriveFrom="Trigger-Index-List">
            <Attribute Id="Trading-Symbol" Domain="Trading-Symbol-List"/>
            <Attribute Id="Entry-Metric" Domain="Metrics"/>
            <Attribute Id="Entry-Ratio" Domain="Metric-Ratio-List"/>
            <Attribute Id="Exit-Metric" Domain="Metrics"/>
            <Attribute Id="Exit-Ratio" Domain="Metric-Ratio-List"/>
            <Attribute Id="Shares"/>
            <Attribute Id="Bought-Price"/>
            <Attribute Id="Sold-Price"/>
            <Attribute Id="Profit"/>
            <Attribute Id="Day-Number"/>
        </DataEntity>
      - <DataEntity Id="Equity-Metric" DeriveFrom="Equity-Metrics">
```


Figure 5-12: Stock-Trading Schema

```
        <Attribute Id="Equity-Metrics.IdentifierId"/>
        <Attribute Id="Equity-Metrics.Metric"/>
        <Attribute Id="Equity-Metrics.MetricValue"/>
        <Attribute Id="Equity-Metrics.DayNumber"/>
    </DataEntity>
- <DataEntity Id="Equity-History" DeriveFrom="Equity-History">
        <Attribute Id="Equity-History.SymbolId"/>
        <Attribute Id="Equity-History.PriceAtOpen"/>
        <Attribute Id="Equity-History.DayNumber"/>
    </DataEntity>
        <!-- All attributes, problems, and entities are automatically generated as expressions and
        available for use as expressions in the database -->
- <Expression Id="Profit" NullExpression="0" Operator="MAXDIFF_POSITIVE">
        <Mapping ParameterName="SalePrice" Attribute="Sold-Price"/>
        <Mapping ParameterName="CostBasis" Attribute="Bought-Price"/>
    </Expression>
- <Expression Id="Metrics" Operator="SELECT_EXTERNAL">
        <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
        <Mapping ParameterName="Attribute"
            Attribute="Olap.TradingDayCalendar.DayNumber"/>
    </Expression>
- <Expression Id="Trading-Days" Operator="SELECT_EXTERNAL">
        <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
        <Mapping ParameterName="Attribute" Attribute="DayNumber"/>
    </Expression>
- <Expression Id="Trading-Symbols" Operator="SELECT_EXTERNAL">
        <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
        <Mapping ParameterName="Attribute"
            Attribute="Olap.DataSampleEquity.TradingSymbolId"/>
    </Expression>
- <Expression Id="Trading-Indices" Operator="SELECT_EXTERNAL">
        <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
        <Mapping ParameterName="Attribute" Attribute="Olap.DataSampleIndex.IndexId"/>
    </Expression>
- <Expression Id="Equity-Metrics" Operator="SELECT_EXTERNAL">
        <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
        <Mapping ParameterName="Attribute"
            Attribute="Dbo.EquityQuoteMetric.QuoteMetricId"/>
    </Expression>
- <Expression Id="Equity-Metrics.Identifier" Operator="SELECT_EXTERNAL">
        <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
        <Mapping ParameterName="Attribute"
            Attribute="Dbo.EquityQuoteMetric.Identifier"/>
    </Expression>
- <Expression Id="Equity-Metrics.Metric" Operator="SELECT_EXTERNAL">
        <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
        <Mapping ParameterName="Attribute"
            Attribute="Dbo.EquityQuoteMetric.MetricId"/>
    </Expression>
```


Figure 5-13: Stock-Trading Schema (Continued)

```
- <Expression Id="Equity-Metrics.MetricValue" Operator="SELECT_EXTERNAL">
      <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
      <Mapping ParameterName="Attribute"
          Attribute="Dbo.EquityQuoteMetric.MetricValue"/>
  </Expression>
- <Expression Id="Equity-Metrics.DayNumber" Operator="SELECT_EXTERNAL">
      <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
      <Mapping ParameterName="Attribute"
          Attribute="Dbo.EquityQuoteMetric.DayNumber"/>
  </Expression>
- <Expression Id="Equity-History" Operator="SELECT_EXTERNAL">
      <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
      <Mapping ParameterName="Attribute"
          Attribute="Dbo.EquityHistory.EquityHistoryId"/>
  </Expression>
- <Expression Id="Equity-History.SymbolId" Operator="SELECT_EXTERNAL">
      <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
      <Mapping ParameterName="Attribute" Attribute="Dbo.EquityHistory.SymbolId"/>
  </Expression>
- <Expression Id="Equity-History.PriceAtOpen" Operator="SELECT_EXTERNAL">
      <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
      <Mapping ParameterName="Attribute"
          Attribute="Dbo.EquityHistory.PriceAtOpen"/>
  </Expression>
- <Expression Id="Equity-History.DayNumber" Operator="SELECT_EXTERNAL">
      <Mapping ParameterName="DataSource" Attribute="TradingOptimizerDb"/>
      <Mapping ParameterName="Attribute" Attribute="Dbo.EquityHistory.DayNumber"/>
  </Expression>
- <Expression Id="Entry-Metric-Limit" Operator="ADD">
      <Mapping Attribute="Entry-Ratio"/>
      <Mapping Attribute="Ratio-Increment"/>
  </Expression>
- <Expression Id="Exit-Metric-Limit" Operator="ADD">
      <Mapping Attribute="Exit-Ratio"/>
      <Mapping Attribute="Ratio-Increment"/>
  </Expression>
- <Expression Id="Subtract-Day" Operator="SUBTRACT">
      <Mapping Attribute="Day-Number"/>
      <Mapping Attribute="-1"/>
  </Expression>
      <!-- Lookup Metric to see if met for trigger, then copy in price at close to generate
      portfolio trade-->
- <Query Id="Trade-Market" OutputAttribute="Day-Number" ColumnName="DayNumber"
  BaseExpression="CURRENT_STEP">
          <!-- Base Expression gets equity metrics for current day-->
      <Lookup ColumnName="IndexId" Expression="Portfolio-Index-Transaction"/>
      <Lookup OutputAttribute="Bought-Price" ColumnName="Bought-Price"
          Expression="Bought-Price"/>
```


Figure 5-14: Stock-Trading Schema (Continued)

```
<Lookup OutputAttribute="Sold-Price" ColumnName="Sold-Price" Expression="Sold-
    Price"/>
<Lookup OutputAttribute="Entry-Ratio" ColumnName="Entry-Ratio"
    Expression="Entry-Ratio"/>
<Lookup OutputAttribute="Exit-Ratio" ColumnName="Exit-Ratio" Expression="Exit-
    Ratio"/>
<Lookup OutputAttribute="Trading-Symbol" ColumnName="Trading-Symbol"
    Expression="Trading-Symbol"/>
<Lookup ColumnName="Metric-Day-Number" Expression="Subtract-Day"/>
    <!-- Lookup prior day metric in order for trigger index to determine if trade should
    be made for next day-->
- <Lookup ColumnName="Quote-Metric-Id" Expression="Equity-
Metrics.QuoteMetricId">
    <Criteria Attribute="Equity-Metrics.DayNumber" CompareOperator="EQUAL"
        QueryColumn="Metric-Day-Number"/>
</Lookup>
- <Lookup ColumnName="Quote-Metric-Id" Expression="Equity-
Metrics.DayNumber">
    <Criteria Attribute="Equity-Metrics.IdentifierId" CompareOperator="EQUAL"
        QueryColumn="IndexId"/>
</Lookup>
    <!-- Identify the unique metric records for the day/index-->
- <Lookup OutputAttribute="Entry-Metric" ColumnName="Entry-Metric-Id"
Expression="Equity-Metrics">
    <Criteria Attribute="Equity-Metrics.QuoteMetricId"
        CompareOperator="EQUAL" QueryColumn="Quote-Metric-Id"/>
        <!-- Get the Entry Metric associated to the triggering index linked to the
        portfolio -->
</Lookup>
- <Lookup ColumnName="Entry-Metric-Value" Expression="Equity-
Metrics.MetricValue">
    <Criteria Attribute="Equity-Metrics.QuoteMetricId"
        CompareOperator="EQUAL" QueryColumn="Quote-Metric-Id"/>
        <!-- Get the Entry Metric Value associated to the triggering index linked to the
        portfolio -->
</Lookup>
- <Lookup OutputAttribute="Exit-Metric" ColumnName="Exit-Metric"
Expression="Equity-Metrics.Metric">
    <Criteria Attribute="Equity-Metrics.QuoteMetricId"
        CompareOperator="EQUAL" QueryColumn="Quote-Metric-Id"/>
        <!-- Get the Exit Metric associated to the triggering index linked to the
        portfolio -->
</Lookup>
- <Lookup ColumnName="Exit-Metric-Value" Expression="Equity-
Metrics.MetricValue">
    <Criteria Attribute="Equity-Metrics.QuoteMetricId"
        CompareOperator="EQUAL" QueryColumn="Quote-Metric-Id"/>
        <!-- Get the Exit Metric Value associated to the triggering index linked to the
        portfolio -->
```


Figure 5-15: Stock-Trading Schema (Continued)

```
    </Lookup>
-   <Lookup OutputAttribute="Bought-Price" ColumnName="PriceAtOpen"
    Expression="Equity-History.PriceAtOpen">
            <Criteria Attribute="Equity-History.DayNumber" CompareOperator="EQUAL"
                QueryColumn="DayNumber"/>
    </Lookup>
-   <Lookup OutputAttribute="Bought-Price" ColumnName="PriceAtOpen"
    Expression="Equity-History.PriceAtOpen">
            <Criteria Attribute="Equity-History.SymbolId" CompareOperator="EQUAL"
                QueryColumn="Trading-Symbol"/>
    </Lookup>
-   <Lookup OutputAttribute="Bought-Price" ColumnName="PriceAtOpen"
    Expression="Equity-History.PriceAtOpen">
            <Criteria Attribute="NULL" CompareOperator="IS" QueryColumn="Bought-
                Price"/>
    </Lookup>
            <!-- Get the Price at Open for Entry - Price will only be used if filter passes and no
            prior bought price (null)-->
-   <Lookup OutputAttribute="Sold-Price" ColumnName="PriceAtOpen"
    Expression="Equity-History.PriceAtOpen">
            <Criteria Attribute="Equity-History.DayNumber" CompareOperator="EQUAL"
                QueryColumn="DayNumber"/>
    </Lookup>
-   <Lookup OutputAttribute="Sold-Price" ColumnName="PriceAtOpen"
    Expression="Equity-History.PriceAtOpen">
            <Criteria Attribute="Equity-History.SymbolId" CompareOperator="EQUAL"
                QueryColumn="Trading-Symbol"/>
    </Lookup>
-   <Lookup OutputAttribute="Sold-Price" ColumnName="PriceAtOpen"
    Expression="Equity-History.PriceAtOpen">
            <Criteria Attribute="NULL" CompareOperator="IS" QueryColumn="Sold-
                Price"/>
    </Lookup>
-   <Lookup OutputAttribute="Sold-Price" ColumnName="PriceAtOpen"
    Expression="Equity-History.PriceAtOpen">
            <Criteria Attribute="NULL" CompareOperator="IS_NOT"
                QueryColumn="Bought-Price"/>
    </Lookup>
            <!-- Get the Price at Open for Exit - Price will only flow out if filter passes and no
            prior Sold price (null)-->
            <!-- Matrix contains all of the metric values for the day for the triggering index.
            Next, filter the metrics using information from the portfolio-index-transaction to
            determine if entry or exit mode and apply the metric value selection parameters-->
    <Filter Id="Exit-Needed" Operator="IS_NOT" Column2="NULL" Column1="Bought-
            Price"/>
    <Filter Id="Equity-Not-Sold" Operator="IS" Column2="NULL" Column1="Sold-
            Price" AndFilter="Exit-Needed"/>
            <!-- Qualifies for selling -->
```


Figure 5-16: Stock-Trading Schema (Continued)

```
                <Filter Id="Entry-Needed" Operator="IS" Column2="NULL" Column1="Bought-
                    Price" OrFilter="Exit-Needed"/>
                    <!-- Qualifies for buying-->
                <Filter Id="Entry-Valid" Operator="GREATER_THAN" Column2="Entry-Ratio"
                    Column1="Entry-Metric-Value" AndFilter="Entry-Needed"/>
                <Filter Id="Entry-Within-Increment" Operator="LESS_THAN" Column2="Entry-
                    Metric-Linit" Column1="Entry-Metric-Value" AndFilter="Entry-Valid"/>
                <Filter Id="Exit-Valid" Operator="GREATER_THAN" Column2="Exit-Ratio"
                    Column1="Exit-Metric-Value" AndFilter="Exit-Needed"/>
                <Filter Id="Exit-Within-Increment" Operator="LESS_THAN" Column2="Exit-
                    Metric-Linit" Column1="Exit-Metric-Value" AndFilter="Exit-Valid"/>
            </Query>
                <!-- -->
        </Problem>
</Upr>
```


Figure 5-17: Stock-Trading Schema (Continued)

```
- <xs:Simulation StartDate="2000-01-03" Id="TreasuryVolatility"
  xmlns:xs="http://tempuri.org/XMLSchema.xsd">
    - <xs:Variable Id="TestRange_MACD_12_26">
        - <xs:ValueFunction TestMethod="RANGE_UNBOUND" Function="VARY_BY_INCREMENT">
            <xs:Argument Value=".98" Name="Start"/>
            <xs:Argument Value="1.02" Name="End"/>
            <xs:Argument Value=".02" Name="Increment"/>
          </xs:ValueFunction>
            <!-- Test for near cross-over as well as completed and lump all greater than 1.02 together
            (range_unbound)-->
      </xs:Variable>
    - <xs:Variable Id="ExitRatio">
        - <xs:ValueFunction TestMethod="RANGE_UNBOUND" Function="VARY_BY_INCREMENT">
            <xs:Argument Value="1.02" Name="Start"/>
            <xs:Argument Value="1.08" Name="End"/>
            <xs:Argument Value=".03" Name="Increment"/>
          </xs:ValueFunction>
            <!-- Test level of greed for profit from 2 percent gain to 8 percent and above-->
      </xs:Variable>
    - <xs:Variable Id="StopRatio">
        - <xs:ValueFunction Function="VARY_BY_INCREMENT">
            <xs:Argument Value=".84" Name="Start"/>
            <xs:Argument Value=".98" Name="End"/>
            <xs:Argument Value=".04" Name="Increment"/>
          </xs:ValueFunction>
            <!-- Test level of caution for loss of 16% through 2%-->
      </xs:Variable>
    - <xs:Variable Id="VaryLookbackIncrement">
        - <xs:ValueFunction TestMethod="EXACT" Function="VARY_BY_INCREMENT">
            <xs:Argument Value="0" Name="Start"/>
            <xs:Argument Value="160" Name="End"/>
            <xs:Argument Value="20" Name="Increment"/>
            <xs:Argument Value="2" Name="Multiplier"/>
          </xs:ValueFunction>
            <!-- Try autolearn intervals of cummulative (0), 20, 40, 80, 160 (Increment Multiplier is 2)-->
      </xs:Variable>
    - <xs:Variable Id="VaryLookbackOffset">
        - <xs:ValueFunction TestMethod="EXACT" Function="VARY_BY_INCREMENT">
            <xs:Argument Value="0" Name="Start"/>
            <xs:Argument Value="80" Name="End"/>
            <xs:Argument Value="20" Name="Increment"/>
            <xs:Argument Value="2" Name="Multiplier"/>
          </xs:ValueFunction>
            <!-- Try autolearn lookback starting from 0, 20, 40, 80 days back-->
      </xs:Variable>
    - <xs:Variable Id="EntryOrderType">
        <xs:Value>MKT</xs:Value>
      </xs:Variable>
    - <xs:Variable Id="ExitOrderType1">
        <xs:Value>LMT</xs:Value>
      </xs:Variable>
    - <xs:Variable Id="ExitOrderType2">
        <xs:Value>STP</xs:Value>
      </xs:Variable>
```

Figure 5-18: CapGen-specific Schema (Variables)

```
    </xs:Variable>
    <xs:DataValue Id="PriorPrice_^VIX" Measurement="PriceAtClose" IntervalOffset="-1" IntervalLength="1"
        Instrument="^VIX"/>
        <!-- Prior day closing price-->
    <xs:DataValue Id="ClosePrice_^VIX" Measurement="PriceAtClose" IntervalOffset="0" IntervalLength="1"
        Instrument="^VIX"/>
        <!-- Last closing price -->
    <xs:DataValue Id="MACD_12_26_ADDN" Measurement="PriceAtClose" IntervalOffset="0" IntervalLength="1"
        Instrument="^ADDN" Metric="MACD_12_26"/>
        <!-- The Cross over divergence of 12/26 day average for ADDN for last trading day -->
    <xs:Condition Id="Condition_TestRange_MACD_12_26" TargetOperator="MULTIPLY"
        TargetDataValue="MACD_12_26_ADDN" TargetVariable="TestRange_MACD_12_26"
        CompareOperator="GREATER" SourceDataValue="MACD_12_26_ADDN"> </xs:Condition>
        <!-- Check if the MACD_12_26 for last trading day met parameter criteria-->
    <xs:Condition Id="Test_VIX_LOWER" TargetDataValue="ClosePrice_^VIX"
        CompareOperator="GREATER_THAN" SourceDataValue="PriorPrice_^VIX"> </xs:Condition>
        <!-- Check if the prior day closing price for the vix was more than today's VIX -->
  - <xs:Workflow>
        <xs:Trigger UnionGroupId="0" Condition="Condition_TestRange_MACD_12_26"/>
        <xs:Trigger UnionGroupId="0" Condition="TEST_VIX_LOWER"/>
        <xs:Parameter TransformOperator="EQUAL" ExecutionAttribute="ORDER_TYPE"
            Variable="EntryOrderType"/>
        <xs:Instrument>TBT</xs:Instrument>
            <!-- Enter at Market if BOTH tests are true (UnionGroupId is the same - Different union groups generate
            OR conditions)-->
    </xs:Workflow>
  - <xs:Workflow>
      - <xs:Parameter UnionGroupId="0" TransformOperator="MULTIPLY" ExecutionAttribute="EXIT_PRICE"
        Variable="ExitRatio">
            <xs:SourceAttribute>FILL_PRICE</xs:SourceAttribute>
        </xs:Parameter>
      - <xs:Parameter UnionGroupId="0" TransformOperator="MULTIPLY" ExecutionAttribute="EXIT_PRICE"
        Variable="StopRatio">
            <xs:SourceAttribute>FILL_PRICE</xs:SourceAttribute>
        </xs:Parameter>
        <xs:Instrument>TBT</xs:Instrument>
            <!-- Test the various exit and stop limits - All combinations will be tested since UnionGroupId is the
            same-->
    </xs:Workflow>
    <xs:Autolearn IntervalOffset="VaryLookbackOffset" IntervalLength="1" Level="1"
        IntervalCount="VaryLookbackInterval"> </xs:Autolearn>
    <xs:Autolearn IntervalOffset="0" IntervalLength="1" Level="2" IntervalCount="0"> </xs:Autolearn>
        <!-- Create the auto-learns for level 1 to evaluate the profit windows and adjust strategy as well as the level 2
        to learn from the lower level autolearns -->
</xs:Simulation>
```

Figure 5-19: CapGen-specific Schema (Values and workflow)

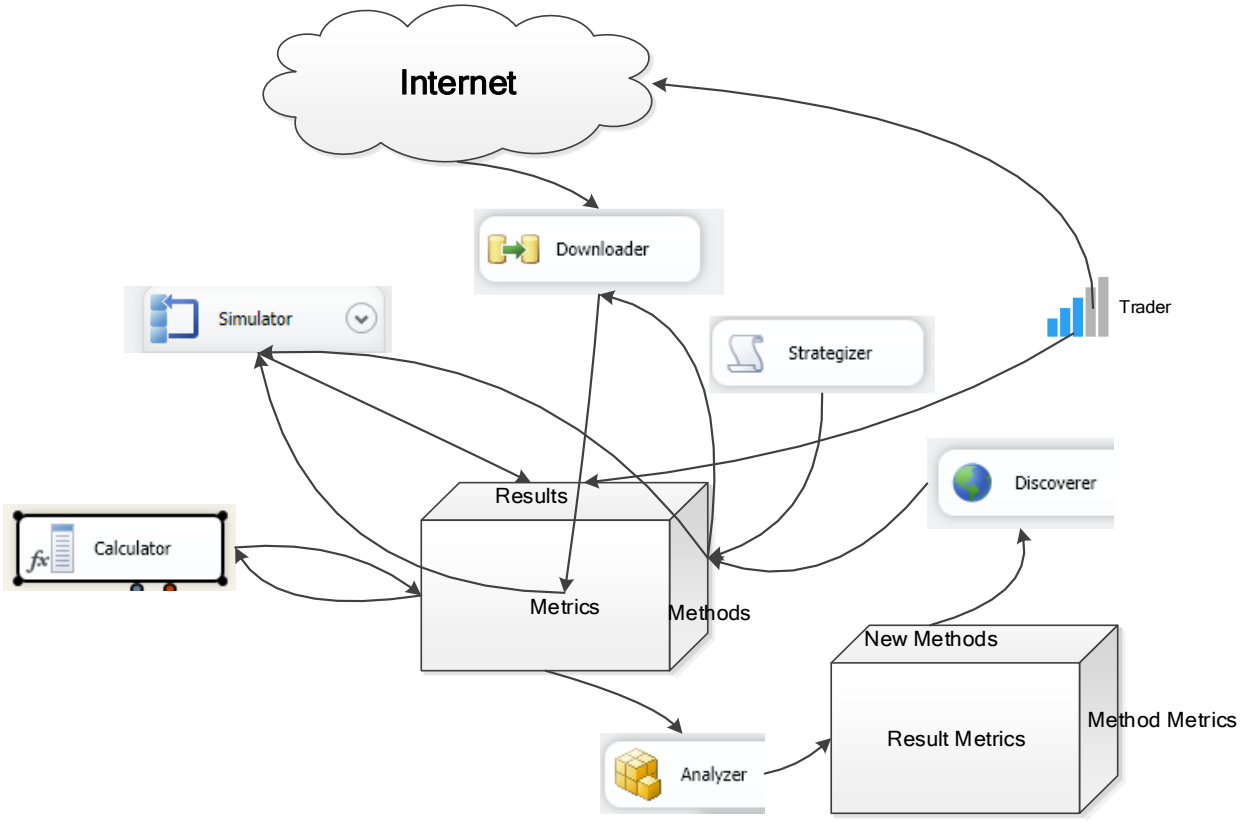


Figure 5-20: Trading System with Feedback Loop and Dynamic Web Data Collection

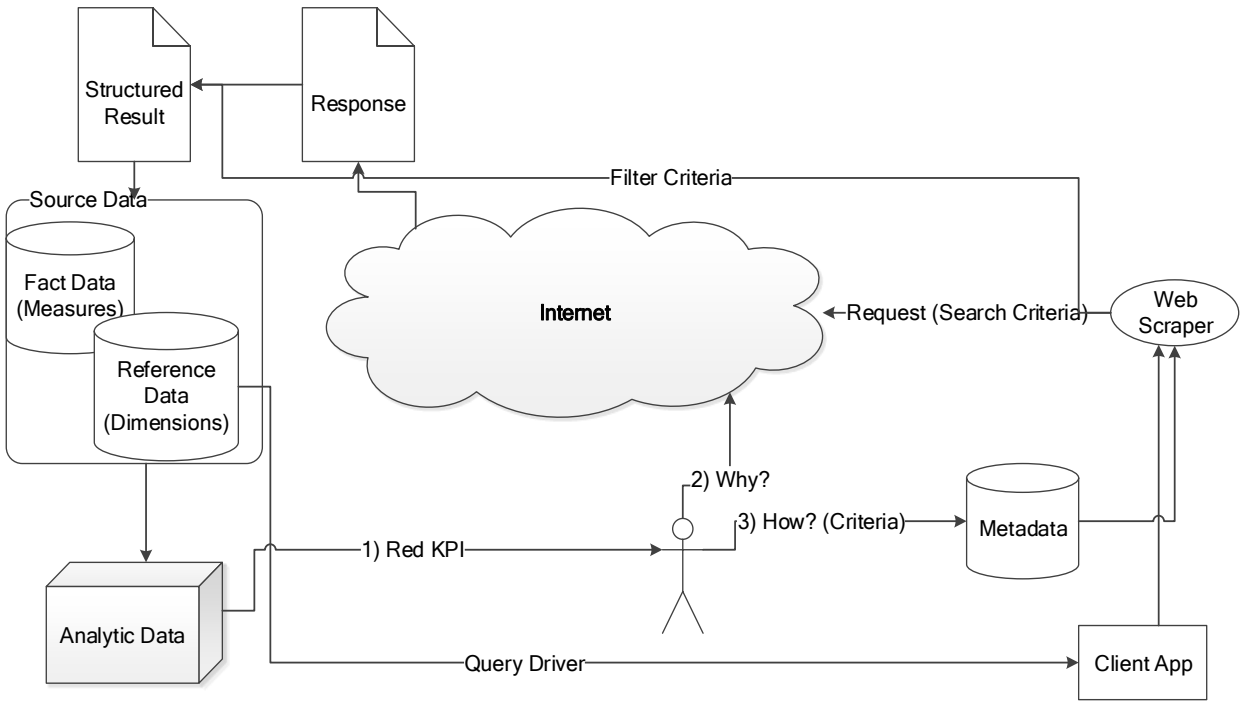


Figure 5-21: Automated Data Capture Scheme

Figure 5-22: Stock Trading Report Showing Impact of AutoLearn for Timing

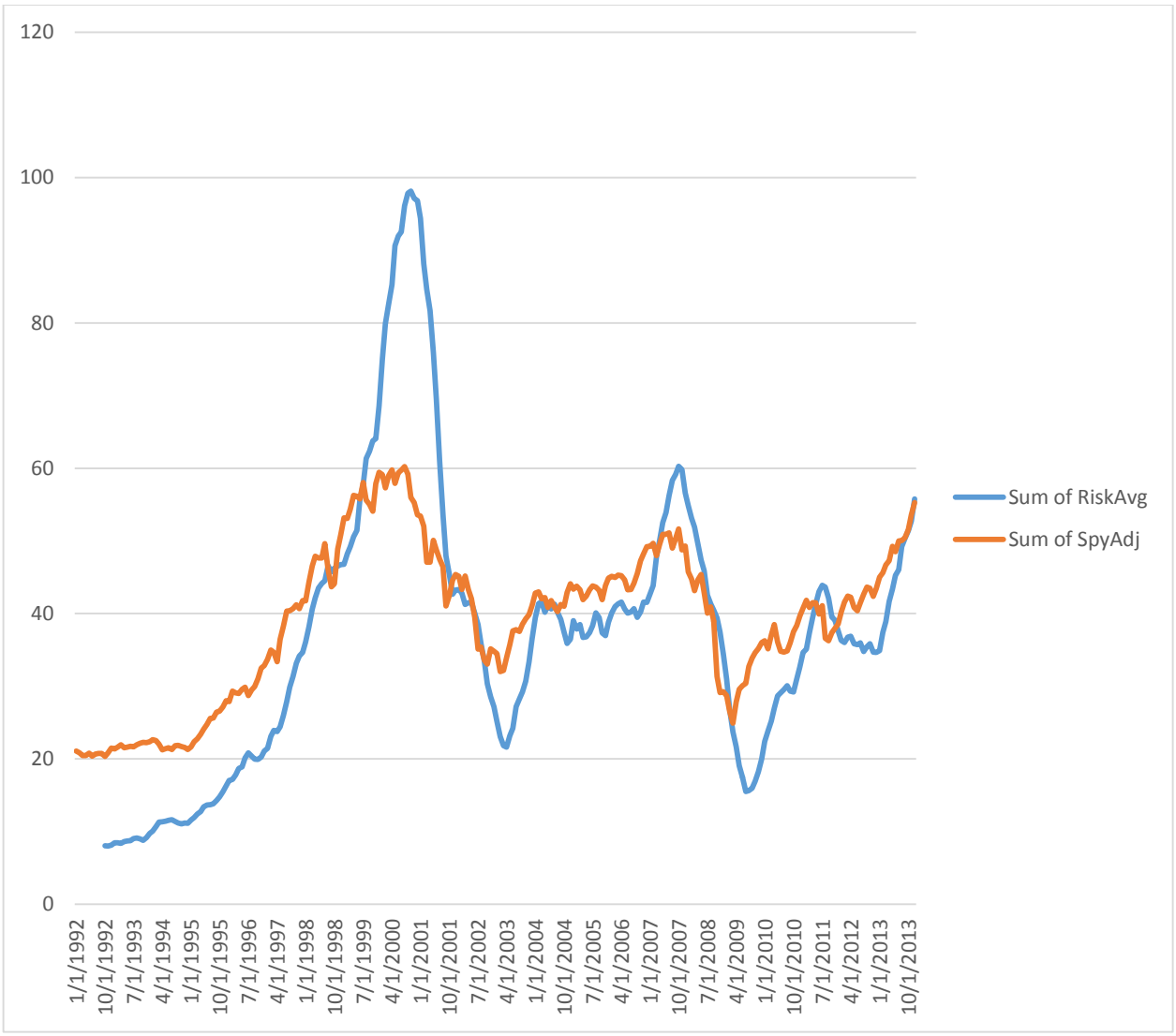


Figure 5-23: Graph showing Predictive Analysis from Simulations

5.2 Implementation Feasibility

Implementation feasibility revolves around multiple aspects of scalability. In this section, the following aspects are considered:

1. Efficiency – What level of efficiency occurs with UPRF? Do the costs support the benefits? Does the learning speed up the solving process at a faster pace than the complexity of the solving endeavors?
2. Architectural Scalability – Does the solution support scaling data and processes? This examines questions such as:
    a. Can the data utilized by the system be distributed and shared among multiple processes?
    b. Can separate processes be instantiated to operate on sub-aspects of a problem-solving endeavor? Can the simulation and exploration of transformation sequences be distributed effectively and coordinated?

5.2.1 Efficiency

Efficiency is an attribute of systems to ensure achieving the maximum functionality with the minimum amount of overhead. The degree that a system can be efficient is highly correlated to not only the overall system architecture, but the sophistication of the algorithms used to carry out the tasks [126]. Both the internal infrastructure for problem solving and the executed operators in problem solving endeavors to recognize transformational patterns determine the efficiency of UPRF. The internal architecture for UPRF is agent-based and distributable while the hosting architecture is extensible to support executing functions as part of the transformation process. In the second regard, this might seem to make UPRF limited in performance to the given algorithms;

however, the internal architecture of the system supports the concept of equilibrium and of schematizing any problem for optimization. This includes the performance of UPRF itself. Therefore, the system promotes efficiency improvement while still allowing extensibility.

The Tower of Hanoi provides a good example of the benefit for the continued optimization through transformation of lower level instances into higher order problems that seek the generic solution patterns. When the framework obtains the generic solution pattern, it drastically reduces the effort for solving further instances in the domain. Although the Tower of Hanoi is itself NP-hard, the solution effort does not need to be brute-force. Although the number of steps to represent the solution increases exponential, once the solution pattern is identified, the number of steps required to find the solution path can be linear. Table 5-13 shows the number of operations associated with both the simulation and transformation instances for Hanoi. Figure 5-24 shows the outcome in numbers of operations. Note that as the transformation increases to higher levels, the number of operations decreases and the exponential correlation to the number of discs disappears. Once the generic solution path is in place, the system only needs to reverse through the transformation process to generate the solution steps directly.

Table 5-13 Solution Complexity

Problem Instance	**Objects**	**Solve Steps**
S1 - Simulation Solution (discs =2)	2	5
S2 - Simulation Solution (discs = 3)	3	39
S3 - Simulation Solution (discs = 4)	4	221
S4 - Simulation Solution (discs = 5)	5	1975
T1 - Transformation Solution (S1>S2)	3	232
T2 - Transformation Solution (S2>S3)	4	495
T3 - Transformation Solution (S3>S4)	5	1055
TS1 - Transformation Sequence Solution (T1>T2)	6	2239
TS2 - Transformation Sequence Solution (T2>T3)	6	2239

SG1 - Sequence Generation Solution	3	3

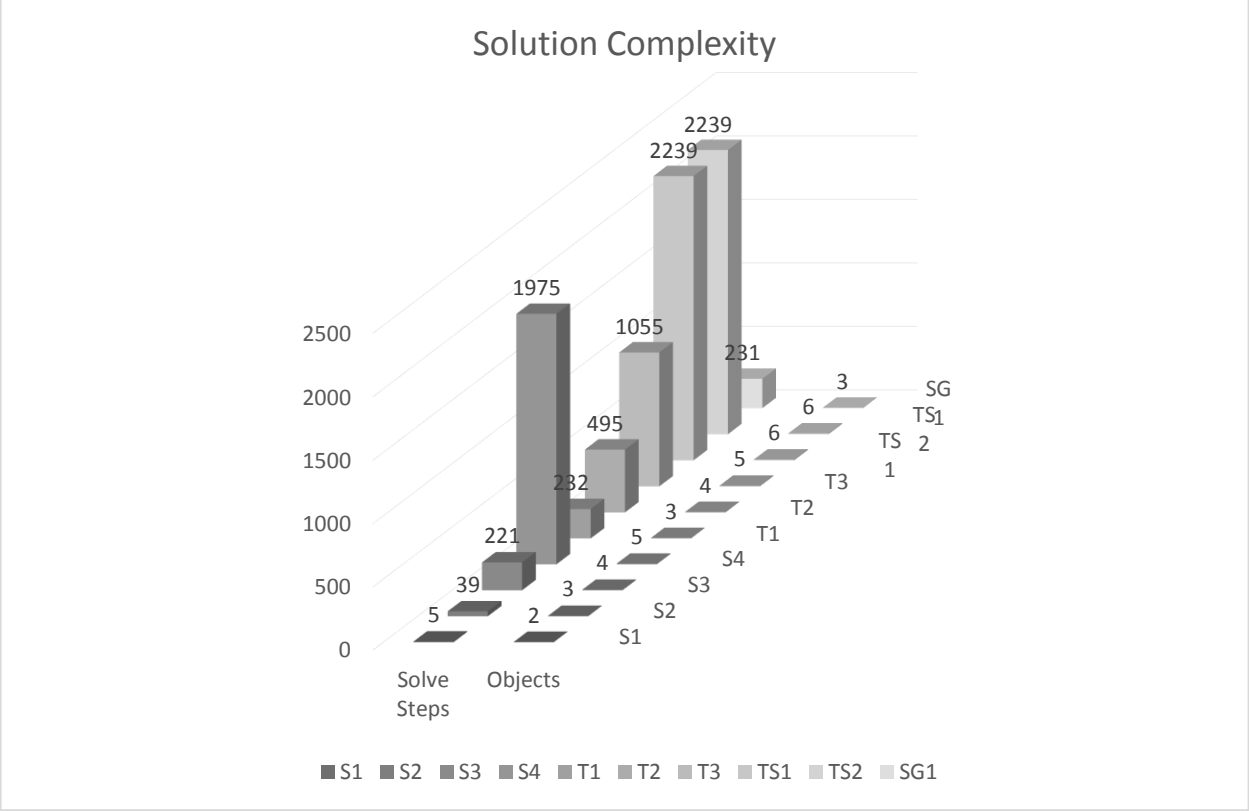


Figure 5-24 Solution Complexity

The identification of the simulation solution for S5 with five discs is therefore reduced to the three steps from the solution generation path which directly generates TS2 which then generates T4 which generates the S5 solution path such that the number of steps is only five to perform the transformation. The identification steps only rely on the prior instance being in place, which the framework generates through the five-step process. Therefore, even though the number of operations required to generate the steps into a representation are exponential with regard to the discs, the time complexity for identifying the transformation and generating the sequence is linear. This achieves the same level of performance as if predesigning an algorithm to determine the

correct moves. This use case demonstrates that, with no pre-knowledge and exclusively dependent on simulation and transformation, UPRF is ultimately able to achieve equivalent performance to an algorithm specifically designed for the problem.

By storing all problem descriptions within a generic structure along with queries that represent the relational states required for the initial state, transitive state, and goal state, UPRF is able to utilize a general-purpose simulation engine and transformer without the need for custom applications for different domains. This provides the ability for autonomous agents to play out the simulations. This, along with the state capture framework, provides complete solution sequences for all problem instances to the transformation engine. This promotes scalability since each agent has the data needed in order to accomplish their task as well as allowing agents to execute on independent nodes. This ability to distribute tasks and utilize independent agents that are task-oriented rather than domain-oriented promotes not only scalability but also higher efficiency [127].

Another area of efficiency concerns minimizing the number of operations required to find the correct sequencing of operations to perform the transformation. The number of possible sequences grows exponentially related to the number of possible operators. This indicates the important of the cross-domain learning aspect in order to weight the testing of operators more likely to be of use in a particular domain based on learning in another domain. The solving engine performance will slow significantly if an abundance of operators requires testing. Transformational operators can perform operations that transform the entire sequence as well as just portions. If a transformational operator emerges that can generate an instance solution from another instance solution in a single operation, this mitigates the exponential nature of the exercise. This dissertation

does not investigate this in detail and this topic along with machine learning optimization in general represents an area for further research.

5.2.2 Architectural Scalability

UPRF scales in multiple directions involving process, data, and tasks. The independent agent architecture supports using different nodes to target specific aspects of the framework such as schematization, simulation, and transformation. The only requirement is that each node have access to the data. Scaling of data is achievable through partitioning of the relational data across nodes. The framework achieves task scalability through the instantiation method whereby each simulation or transformation endeavor relates to a particular instance of a problem and is self-contained. Once again, the only requirement is that all operations are able to register with a central data store and receive information from the data store. Even within each problem-solving instance, each operation is step is a new instance and interacted with by each agent at the step level. Figure 5-25 illustrates the method in which UPRF scales out across agents, data, and tasks with sub-instances.

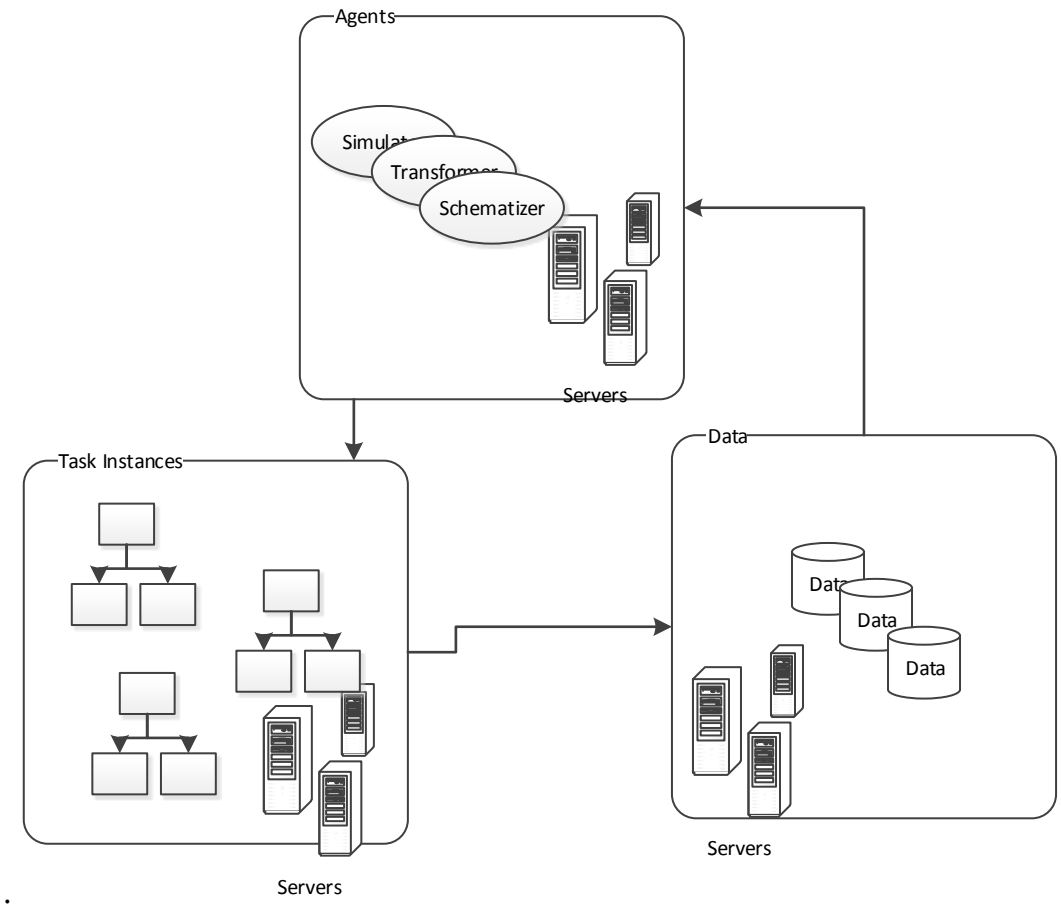


Figure 5-25: Scalability Model for UPRF

5.3 Chapter Summary

This chapter focused on the practicality of a universal problem resolution framework (UPRF). To be of relevance for further study, the framework must show some promise to be usable within a foreseeable time horizon given current and emerging computational capabilities. This chapter dealt with concerns related to problem solving, since it is the same type of challenge as that presented by automated theorem proving. Therefore, problem solving is NP-Complete in nature and exponential, which implies limited utility. The chapter introduced other problems of different types, potentially much larger and more complex than the Tower of Hanoi base use case including

a demonstration of the extensibility of UPRF to handle increasing complexity for an existing scenario (transposing the 3-peg Tower of Hanoi to a K-peg problem). Despite these challenges, UPRF promises to be practical because of its ability to scale both the scope of problems and the use of computational resources efficiently. The key driving force is the inherent capability within the framework to model its own performance and continually learn from it in order to achieve the equilibrium goal of maximum benefit within cost constraints.

6. CONCLUSION AND FURTHER RESEARCH

This section evaluates the results of the operational proof as well as contributions from the other chapters that outline the supporting literature, design approach, and practicality for applications against the research questions and goals. In order to answer the research goals, the following outcomes were indicated:

1) Proof by example that a continuous improvement universal problem resolution framework can be constructed using currently available software and hardware;

2) Production of a prototype that meets the thesis for a continuous improvement system – i.e. non-domain specific, extensible without reprogramming of the core system, lacking need of subject matter expert intervention; and executing within polynomial time complexity;

3) Rationale that the framework coupled with technology advancements generates optimal solutions using fewer resources than possible without such a framework.

The following research questions were posited in the introduction and evaluated in terms of the outcomes.

1) **Feasibility**: Is it possible to build a universal problem resolution system that continuously improves and learns from its own execution including automatically cross applying concepts learned to other domains in the most optimal method possible given the information available to the system?

2) **Construction**: What is the design of such a universal problem resolution framework (UPRF) described above? What are the constraints and requirements for a design that fully supports generic representation of problems, generic pursuit of problem solutions and continuous improvement utilizing an overarching set of processing components **without** the need for modifications of the actual components for the solving system? The specific subordinate questions under this were:
    a. How can UPRF generically encode problems from any domain without the need for redesign of the problem representation process? What is the level of effort required to represent various types of problems?
    b. How can UPRF utilize the generic representation of a problem to explore possible solutions?
    c. How can UPRF support a continuous improvement paradigm?
3) **Practicality**: How would UPRF model, solve, and improve various types of problems? What Subject-Matter-Expert (SME) interactions are required? How are these interactions minimized? Is UPRF practical given current and potential technology?

6.1 Results

6.1.1 Feasibility

Chapter 3 provided a framework for storing problems generically while chapter four provided an operational proof of UPRF using the NP-hard problem the Tower of Hanoi as the use case. The operational proof demonstrated that the framework could transform sequences for different instances generated from a deterministic problem to a more generalized problem for finding the sequences. Through proof-by-example, the framework created higher order transformations while

solving more instances up to four levels, at which time a generalized pattern materialized that reversed to generate the input for the lower level problem transformation. This shows that it is possible not only to generate higher order problems into a common schema but also to probe for solutions to the higher order problems and recursively apply the solution sequences back to the lower order problems. The dissertation thoroughly addressed the challenge of representing a problem through a schema through a relational schema using not only the Tower of Hanoi, but also the transformation problem itself. Chapter 5 on practicality provided additional examples of problem schemas that the same generic schema could represent.

An assertion from the research question concerned the NP-complete challenge. In using an NP-hard scenario, it was obviously impossible to solve a problem in polynomial time even with discovery of the optimal solution. However, the goal of the NP scenario was to demonstrate that the process of solution discovery could be constrained to polynomial time through an equilibrium construct and that the endeavors of the solution discovery could make progress toward optimization of NP problems. While the dissertation did not thoroughly explore this, testing proved that the process of applying the higher order transformation in order to generate a solution instance without the need to explore the full problem space in a simulative fashion contributed only a linear constant to the overall problem solving process.

Another assertion involved cross-domain learning. Although not proven directly, it is possible to infer that, since the framework captures all of the correlations in one problem-solving endeavor, the framework could treat a problem in another domain as an additional instance for a transform operator such that a correlation leads to an optimization across the domains. A K-peg Hanoi problem, which included simulating solutions to four and five pegs, provided an example to

demonstrate this potential capability. This example verified how learning of the transform sequence for the three-peg solution contributes toward an optimal solution for four pegs. Other examples include executing the learning process against different scenarios to see how optimizations learned in checkers apply to chess. These type of scenarios are useful for understanding the problem crossover capabilities of UPRF and validate its utility for general problem solving.

6.1.2 Construction

Chapter 4 demonstrated the construction of UPRF. Much of the construction utilizes actual working code, although at the time of this work only the simulation and sequence generation were fully working. The UPD problem was not fully implemented and operational for generating transformations and the operational proof required manual calculation to perform the transformations. However, given the ability to manually generate the transformations and verify their correctness through the reversal process, it is feasible to create UPRF. Although UPRF was incomplete in this work, enough of it was prototyped to verify the concepts and establish the key design principles. The three main sub-questions concerned (1) the ability to represent problems generically, (2) the ability to generically search for solutions, and (3) the ability to learn continuously – all of which the operational proof of Chapter 4 verified.

6.1.3 Practicality

The dissertation successfully modeled other applications including collaborative and cooperative involving multiple players or agents for UPRF in Chapter 5. It also discussed the handling of multi-agent scenarios with multiple competing as well as collaborative scenarios. The dissertation examined the problem of modelling and optimizing NP-complete problems. It also

modelled non-deterministic problems such as stock market trading. In all cases, state change capture of individual values across attributes was practical in order to generate higher order transformation problems as candidates for sequence recognition by higher order transformation analysis.

The massive data tracking requirements for UPRF are realizable given the technological trends. What was once thought impossible is now enabled through PCIE SSD [128]; very fast high-speed storage scales into the hundreds of terabytes and distributed computing enables memory to be treated as storage, allowing a network of computers to scale over high-speed networks into the petabytes of storage operating at the speed of memory. Trends in data mining, and predictive analytics using high-speed clusters and technologies such as Hadoop and H-Base are also supportive of the high-performance computing requirements needed by this solution [129]. UPRF's agent-based and task-oriented architecture is able to leverage current and emerging technologies. This includes distributed problem solving techniques such as MapReduce [130] whereby a network of computers supports distributed problem solving through reduction into sub-problems that map onto computational nodes. PCIE SSD or in-memory databases have the potential to support the large amount of data needed for the state representations.

GPU technologies also enable high-speed distributed processing [131]. Chapter 3 provided a system framework based on a scalable broker architecture to allow dispatching of tasks for parallel execution at a very low granularity. The simulation architecture leveraging parallel branching scales very well to make UPRF practical.

6.2 Further Research

This dissertation focused mainly on the modelling, simulation, sequence capture, and transformational mechanism to generate higher order transformation problems from related instances. The work focused on the software framework to support this, rather than the implementation of the system. Many unanswered questions and future endeavors spawn from the framework concepts. This includes areas of machine learning research, in-depth research into specific types of problem domains, and the actual work of constructing the system.

6.2.1 Machine Learning

This work did not address the construction and evolution of the transformation operator functions required to interpret solution sequences associated with one instance and generate a related instance solution through correlation and machine learning. Machine learning was touched upon briefly as a black box that receives the sequence from one instance and then attempts to find the correlations needed to predict the sequence for another instance related through some difference in variable value between the instances. The framework design supports the dynamic addition of functions (transform operators) that perform transformational inspection to generate a target sequence from a source sequence. The integration of machine learning systems as black boxes will enhance the advancement of UPRF for practical solving of real world problems. An interesting possibility enabled by this research is the transposition of UPRF as an actual machine-learning algorithm in and of itself.

6.2.2 Problem Domain Research

This work covered multiple domains but did not go into depth regarding any problem domains other than the Tower of Hanoi example. The work showed through diverse varieties of problems

including NP-complete, it is possible to schematize non-deterministic, probabilistic, and multi-agent and combinations of problems with these attributes for a simulatory approach to solve instances and generate higher order transformational problems to converge instance solutions. Deeper studies into these other realms will enhance the area of automated problem solving and automated solution optimization. This research could result in greater optimizations of all types of problems as well as advance the feasibility for the all-encompassing generic problem system envisioned by this work. Additionally, deep endeavors into automatic optimization of various problem domains have the potential to generate cross-domain knowledge that is applicable to various and diverse problem domains that are only remotely correlated.

6.2.3 Construction of UPRF

This dissertation provided a prototype of the software and code involved in building the framework. Although the dissertation provided code only for the database modelling and schema, it did put forth sufficient design to build the functional code and provided a manual walk-through of the procedural code requirements. The work resulted in code for functions sufficient to automate the simulation and sequence generation, but there was insufficient time to finish development and testing of the code for the transformational aspect. The work did however put forward extensive design concepts for UPRF. For the framework to become a reality, it must implement the design concepts in code within the context of a software development effort. The framework outlined for autonomous problem solving supports an iterative approach so that it is possible to add functionality incrementally in specific problem domains. There is also the potential to bridge existing systems such as that depicted with the stock market optimization scenario to provide immediate benefit to already developed problem solving or optimization applications.

6.3 Chapter Summary

This chapter provides the conclusion to the dissertation, outlining results and further research. The results demonstrate that a universal problem resolution framework (UPRF) is feasible, constructible, and practical. Through the Tower of Hanoi operational proof and the discussion of applications to other problems, UPRF emerges as a novel approach and paradigm shift affecting machine learning and artificial intelligence. The dissertation has shown that UPRF is a viable approach for a problem resolution framework that supports autonomous continuous improvement with minimal external interactions and without the need for ongoing modification of the overall architecture. Further, this work provides an impetus and foundation for other areas of research that can contribute to the domain of generic problem solving systems and be the catalyst for major innovations in future A/I and machine learning frameworks.

7. REFERENCES

[1] F. Buschmann, "On Architecture Styles and Paradigms," *Software, IEEE* , vol.27, no.5, pp. 92-94, Sept. 2010.

[2] A. L. Brown and J. S. De Loache, "Skills, plans, and self-regulation," in *Children's Thinking: What Develops?*, R. S. Siegler, Ed.,. Hillsdale, NJ: Lawrence Erlbaum Assoc., 1978, ch. 1, pp. 3-35.

[3] E. B. Bonawitz *et. al*, "Modeling cross-domain causal learning in preschoolers as Bayesian inference" in *Proc. 28th Annu. Conf. Cognitive Science Society*, Mahwah, NJ, 2006, pp. 89-94.

[4] D. A. Williams et. al, "Configural and elemental strategies in predictive learning," *J. Experimental Psychology: Learning, Memory, and Cognition*, vol. 20, no. 3, pp. 694-709, May 1994.

[5] M. Minsky, "ACM Turing Lectures: Form and Content in Computer Science," *J. of the Assoc. for Computing Machinery*, vol. 17, No. 2, pp. 197-215, Apr. 1970.

[6] A. Wray, "The puzzle of language learning: From child's play to 'linguaphobia,'" *Language Teaching*, vol. 41, no. 2, pp. 253-271, Apr. 2008.

[7] M. C. Corballis, *The Recursive Mind: The Origins of Human Language*,. Princeton, NJ: Princeton University Press, 2011.

[8] A. H. Maslow, "A theory of human motivation," *Psychological Review*, Vol. 50, No. 4, pp. 370-96, 1943.

[9] Merriam-Webster. (2012, November 20). Technology – Definition and More [Online]. Available: http://mw1.merriam-webster.com/dictionary/technology.

[10] J. K. Ousterhout, "Scripting: Higher level Programming for the 21st Century," *IEEE Computer*, vol. 31, no. 3, pp. 23-30, Mar. 1998.

[11] J. N. Foster, "Bidirectional Programming Languages," Ph.D. dissertation, Dept. Comp. and Inf. Science, Univ. of Pennsylvania, Philadelphia, PA, 2009.

[12] J. I. Hungate, *et al.*, "Application Portability Profile and Open System Environment User's Forum," *J. of Research – Nat. Inst. of Standards and Technology*, vol. 100, no. 6, pp. 699-710, Nov. 1995.

[13] M. Rys, "XML and relational database management systems: inside Microsoft® SQL Server™ 2005," in *Proc. 2005 ACM SIGMOD Int. Conf. Management of Data*, New York, NY, 2005, pp. 958-962.

[14] A. Beaulieu, "A Little Background," in *Learning SQL*, 2nd ed. Sebastapol, CA: O'Reilly, 2009, ch. 1, pp. 1-11.

[15] S. Dězeroski, "Data mining in a nutshell," in *Relational Data Mining*. New York: Springer-Verlag New York, 2001, ch. 1, pp. 3-27.

[16] Encyclopædia Britannica. (2012, November 20). *Data mining* [Online]. Available: http://www.britannica.com/EBchecked/topic/1056150/data-mining.

[17] R. Battiti and M. Brunato, "Reactive Business Intelligence: From Data to Models to Insight." Trento, Italy: Srl, 2011.

[18] J. Yue *et al.*, "Predictive Analytics Using a Blackboard-Based Reasoning Agent," in *Int. Conf. Web Intelligence and Intelligent Agent Technology*, 2010, pp.97-100.

[19] X. Chen and I. Petrounias, "A framework for temporal data mining," in *Database and Expert Systems Applications*, Heidelberg, Germany: Springer, 1998, pp. 796-805.

[20] M. E. Toplak and K. E. Stanovich, "The domain specificity and generality of disjunctive reasoning: Searching for a generalizable critical thinking skill," *J. of Educational Psychology*, vol. 94, no. 1, pp. 197-209, Mar. 2002.

[21] J. T. Lanman, "A Governance Reference Model for Service-oriented Architecture-based Common Data Initialization: A Case Study of Military Simulation Federation Systems," Ph.D. dissertation, Dept. of Modeling and Simulation, Univ. of Central Florida, Orlando, FL, 2010.

[22] A. M. Campbell, *et al.*, "Deep blue," *Artificial Intell.*, vol. 134, no. 1, pp. 57-83, Jan. 2002.

[23] J. McCarthy, "AI as Sport," *Science*, vol. 276 no. 5318, pp. 1518-1519, Jun. 1997.

[24] R.S. Hartati, M.E. El-Hawary, "Optimal active power flow solutions using a modified Hopfield neural network," in *Canadian Conf. Elect. and Comput. Eng.*, 2001, pp.189-194.

[25] Chin-Chen Chang *et al.*, "Using Dynamic Programming Strategy to Find an Optimal Solution to Exploiting Modification Direction Embedding Method," in *3rd Int. Conf. Intelligent Inform. Hiding and Multimedia Signal Process.*, 2007, pp. 26-28, 489-492.

[26] E.T. Chu, *et al.*, "An Optimal Solution for the Heterogeneous Multiprocessor Single-Level Voltage-Setup Problem," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 11, pp. 1705-1718, Nov. 2009.

[27] S.O. Orero, M.R. Irving, "A genetic algorithm modelling framework and solution technique for short term optimal hydrothermal scheduling," *IEEE Trans. Power Syst.*, vol. 13, no. 2, pp. 501-518, May 1998.

[28] C.V. Peroni, *et al.*, "Optimal control of a fed-batch bioreactor using simulation-based approximate dynamic programming," *IEEE Trans. Control Syst. Technol.*, vol. 13, no. 5, pp. 786- 790, Sept. 2005.

[29] H. Rotstein and B. Shklyar, "Optimal solution and approximation to the l1/H∞ control problem," *IEEE Trans. Autom. Control*, vol. 44, no. 6, pp. 1240-1243, Jun. 1999.

[30] D. M. Smith, "Overcoming the challenges to feedback-directed optimization (Keynote Talk)," in *Proc. ACM SIGPLAN Workshop Dynamic and Adaptive Compilation and Optimization*, New York, NY, 2000, pp. 1-11.

[31] 1995. Book review: The Implementation of Prolog By Patrice Boizumault Translated from the original French by Ara M. Djamboulian and Jamal Fattouh (Princeton University Press, 1993). *SIGART Bull.* 6, 4 (October 1995), 17-. DOI=10.1145/222267.1065836

[32] Sanner, Michel F. "Python: a programming language for software integration and development." *J Mol Graph Model* 17, no. 1 (1999): 57-61.

[33] O. Goldreich, "On Teaching the Basics of Complexity Theory," in *Lecture Notes in Comput. Sci.*, vol. 3895. Heidelberg, Germany: Springer-Verlag, 2006, pp. 348-374.

[34] G. Gutin and A.P. Punnen, *The Traveling Salesman Problem and Its Variations*, New York, NY: Springer, 2002.

[35] O. Goldreich, "On Teaching the Basics of Complexity Theory," in *Lecture Notes in Comput. Sci.*, vol. 3895. Heidelberg, Germany: Springer, 2006, pp. 348-374.

[36] S. Williams, *The Associative Model of Data*, 2nd ed., Great Britain: Lazy Software, 2000.

[37] E. Meijer and G. Bierman, "A co-Relational Model of Data for Large Shared Data Banks," *Queue*, vol. 9, no. 3, pp. 30-49, Mar. 2011.

[38] B. Whitworth, "Some Implications of Comparing Brain and Computer Processing," *Proc. 41st Annual Hawaii Int. Conf. Syst. Sci.*, Waikoloa, HI, 2008, p. 38

[39] S. Haykin, "Cognitive machines," in *Proc. IEEE Int. Workshop on Mach. Intell. & Signal Process*, Mystic, CT, 2005, pp. 1-20.

[40] S. Haykin, "Cognitive Radar: a way of the future," *Signal Processing Magazine, IEEE*, vol. 23, no.1, pp.30-40, Jan. 2006.

[41] A. M. Turing, "Computing Machinery and Intelligence," *Mind*, vol. 59, no. 236, pp. 433-460, Oct. 1950.

[42] M. Conrad and A. Rosenthal, "Limits on the computing power of biological systems." Bulletin of Mathematical Biology," *Bulleting of Math. Biology*, vol. 43, no. 1, pp. 59-67, 1981.

[43] S. Tschoke, *et al.*, "Solving the traveling salesman problem with a distributed branch-and-bound algorithm on a 1024 processor network," in *9th Int. Proc. Parallel Process. Symp., 1995.*, Shanghai, China, 1995, pp. 182-189.

[44] C.P. Gomes and R. Williams, “Approximation algorithms,” in *Search Methodologies*, 1st ed. New York: Springer, 2005, ch. 18, pp. 557-585.

[45] D. L. Applegate, *et al.*, *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ: Princeton Univ. Pr., 2006.

[46] J. N. MacGregor and T. Ormerod, “Human performance on the traveling salesman problem,” *Perception & Psychophysics*, vol. 58, no. 4, pp. 527-539, 1996.

[47] A.M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, pp. 433-460, Oct. 1950.

[48] A. P. Saygin, *et al.*, “Turing test: 50 years later,” *Minds and Machines*, vol. 10, no. 4, pp. 463-518, Nov. 2000.

[49] Y. Seirawan, *et al.*, “The implications of Kasparov vs. Deep Blue,” *Commun. of ACM*, vol. 40, no. 8, , pp. 21-25, Aug. 1997.

[50] K. L. Sakai, “Language acquisition and brain development,” *Science*, vol. 310, no. 5749, pp. 815-819, Nov. 2005.

[51] S. I. Robertson, “Introduction to the study of problem solving” in *Problem Solving*. East Sussex, England: Psychology Press, 2001, ch. 1, pp. 2-14.

[52] S. A. Kripke, “Semantical Considerations on Modal Logic,” *Acta Philosophica Fennica*, vol. 16, pp. 83-94, 1963.

[53] R. M. Leithiser, “A relational database model for representation of formal specifications,” in *Proc. 44th Annu. ACM Southeast Regional Conf.*, New York, NY, 2006, pp. 209-217.

[54] C. H. Bennett, "Logical reversibility of computation," *IBM J. of Research and Develop.*, vol. 17, no. 6, pp. 525-532, 1973.

[55] K. Gordon, “What is Big Data?” *ITNOW*, vol. 55, no. 3, pp. 12-13, 2013.

[56] M. Mohri, *et al.*, *Foundations of Machine Learning*. Cambridge, MA: MIT Press, 2012.

[57] B. Boehm and L. G. Huang, "Value-based software engineering: a case study," *Computer*, vol.36, no.3, pp.33,41, Mar 2003.

[58] SparkNotes Editors. (2013, September 1). *SparkNote on What is Recursion?* [Online]. Available: http://www.sparknotes.com/cs/recursion/whatisrecursion/

[59] U. Hensel, and B. Jacobs, “Proof principles for datatypes with iterated recursion,” in *Category Theory and Computer Science*. Berlin, Germany: Springer, 1997, pp. 220-241.

[60] L.C. Paulson, “Co-induction and co-recursion in higher order logic,” Univ. of Cambridge Comput. Laboratory, Cambridge, United Kingdom, Rep. 304, 1993.

[61] J. Gibbons and G. Hutton, “Proof methods for corecursive programs,” *Fundamenta Informaticae*, vol. 66, no. 4, pp. 353-366, 2005.

[62] S. Maniccam, "Towers of Hanoi related problems," *J. Comput. Sci. in Colleges*, vol. 27, no. 5, pp. 205-213, May 2012.

[63] D. S. Robertson, "Feedback theory and Darwinian evolution," *J. Theoretical Biology*, vol. 152, no. 4, pp. 469-484, Oct. 1991.

[64] M.M. Lehman, "Feedback in the software evolution process," *Inform. and Software Technology*, vol. 38, no 11, pp. 681-686, Nov. 1996.

[65] I. Thon, *et al.*, "A simple model for sequences of relational state descriptions," in *Machine Learning and Knowledge Discovery in Databases*, Berlin, Germany: Springer, 2008, pp. 506-521

[66] W. Shen and B. Leng, "A metapattern-based automated discovery loop for integrated data mining-unsupervised learning of relational patterns," IEEE Trans. Knowl. Data Eng., vol. 8, no. 6, pp. 898-910, Dec. 1996.

[67] D. A. Naumann, "Ideal models for pointwise relational and state-free imperative programming, in *Proc. 3rd ACM SIGPLAN Int. Conf. Principles and Practice of Declarative Programming*, Florence, Italy, 2001, pp. 4-15.

[68] L. H. Tsoukalas and R. E. Uhrig, *Fuzzy and Neural Approaches in Engineering*, 1st ed.New York, NY: John Wiley & Sons, Inc., 1996.

[69] E. Schikuta, *et al.*, "A Relational Neural Network Database Model," in *Int. ICSC/IFAC Symp. Neural Computation*, Vienna, Austria, 1998, pp. 937-943.

[70] W. Uwents and H. Blockeel, "Classifying relational data with neural networks," in *Inductive Logic Programming*, Berlin, Germany: Springer, 2005, pp. 384-396.

[71] E. Kitzelmann, Emanuel and U. Schmid, "Inductive synthesis of functional programs: An explanation based generalization approach," *J. Mach. Learning Research*, vol. 7, pp. 429-454, Dec. 2006.

[72] P. Langley and D. Choi, "Learning recursive control programs from problem solving," *J. Mach. Learning Research*, vol. 7, pp. 493-518, Dec. 2006.

[73] C. C. McGeoch, "Feature Article-Toward an Experimental Method for Algorithm Simulation," *INFORMS J. Computing*, vol. 8, no. 1, pp. 1-15, Jan. 1996.

[74] A. G. Hoffmann, "On computational limitations of neural network architectures," in *Proc. IEEE Symp. Parallel and Distributed Process*, Dallas, TX, 1990, pp. 818-825.

[75] M. Mitchell, *An Introduction to Genetic Algorithms*, Cambridge, MA: MIT Press, 2006.

[76] D. Sharma, *et. al*., "A domain-specific crossover and a helper objective for generating minimum weight compliant mechanisms," in *Proc. 10th Annu. Conf. Genetic and Evol. Computation*, New York, NY, 2008, pp. 1723-1724.

[77] A. Rogers and A. Prügel-Bennett, "Genetic drift in genetic algorithm selection schemes," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 298-303, Nov. 1999.

[78] E. B. Bonawitz, *et. al.*, "Modeling cross-domain causal learning in preschoolers as Bayesian inference," in *Proc. 28th Annu. Conf. Cognitive Sci. Soc.*, Mahwahe, NJ, 2006, pp. 89-94.

[79] Vasant Dhar. 2013. Data science and prediction. *Commun. ACM* 56, 12 (December 2013), 64-73. DOI=10.1145/2500499 http://doi.acm.org/10.1145/2500499

[80] V. Ranadivé and K. Maney, *The Two-Second Advantage: How We Succeed by Anticipating the Future--Just Enough*. New York: Random House, Inc., 2011.

[81] I. H. Brivanlou, *et al.*, "Mechanisms of Concerted Firing among Retinal Ganglion Cells," *Neuron*, vol. 20, no. 3, pp. 527-539, Mar. 1998.

[82] G. Moreno and V. Pascual, "Programming with fuzzy logic and mathematical functions," in *Proc. 6th Int. Conf. Fuzzy Logic and Applications*, Crema, Italy, 2005, pp. 89-98.

[83]Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. 2012. Simple and practical algorithm for sparse Fourier transform. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA '12). SIAM 1183-1194

[84] L. Zhang, *et al.*, "Self-organization simulation and output analysis," *2000 IEEE Int. Conf. Syst., Man, and Cybern.*, Nashville, TN, 2000, pp. 603-608

[85] R. Holzer and H. de Meer, "On modeling of self-organizing systems," in *Proc. 2nd Int. Conf. Autonomic Computing and Commun. Syst.*, Turin, Italy, 2008, pp. 1-6.

[86] M. F. Salmi and S. Parsa, "Automatic Detection of Infinite Recursion in AspectJ Programs" in *Future Generation Inform. Technology. 1st Int. Conf.*, Jeju Island, Korea, 2009, pp. 190-197.

[87] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. 2012. Simple and practical algorithm for sparse Fourier transform. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA '12). SIAM 1183-1194

[88] E. Romanelli and M. L. Tushman. "Organizational transformation as punctuated equilibrium: An empirical test." *Academy of Manage. J,* vol. 37, no. 5, pp. 1141-1166, Oct. 1994.

[89] A. R. Admati, *et. al.*, "Large shareholder activism, risk sharing, and financial market equilibrium," *J. Political Econ.*, vol. 102, no. 6, pp. 1097-1130, 1997.

[90] Y.S. Chen, *et. al.*, "Mathematical and computer modelling of the Pareto principle," *Math. and Comp. Modelling*, vol. 19, no. 9, pp. 61-80, May 1994.

[91] G. Chalkiadakis and C. Boutilier, "Coordination in multiagent reinforcement learning: a Bayesian approach," in *Proc. 2nd Int. Joint Conf. Autonomous Agents and Multiagent Syst.*, Melbourne, VIC, 2003, pp. 709-716.

[92] J. Moon, "Learning through reflection," in *Early Professional Development for Teachers*, London, England: David Fulton Pub., 2001, ch. 28, pp. 364-378.

[93] L. J. Stockmeyer, "The polynomial-time hierarchy," *Theoretical Comp. Sci.*, vol. 3, no. 1, pp. 1-22, Oct. 1976.

[94] G.F. Smith, "Defining real world problems: a conceptual language," *IEEE Trans. Syst. Man Cybern.*, vol. 23, no. 5, pp. 1220-1234, Sep./Oct. 1993.

[95] A. Chatterjee, "Case Based Reasoning with State Transition Mechanism for Problem-Solving in AI," in *IMECS*, 2006, pp. 346-352.

[96] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377-387, Jun. 1970.

[97] I. J. Rudas and J. Fodor, "Intelligent systems," *Int. J. Comput., Commun. & Control*, vol. 3, pp. 132-138, 2008.

[98] S. Iyengar, "Cognitive Primitives for Automated Learning," in *Proc. 1st AGI Conf. Artificial General Intell.*, Memphis, TN, 2008, pp. 409-413

[99] D. Hanna, "Combinatory adaptive optimization with multi-agent systems," Ph.D. dissertation, Dept. Mech. Eng., Carnegie Mellon Univ., Pittsburgh, PA, 2010.

[100] G. Smith, *et al.*, "Problem Solving System and Method," U.S. Patent 7 885 906, Feb. 8, 2011

[101] A. J. Ramsbotham, "Collective intelligence: toward classifying systems of systems," in *Proc. 9th Workshop Performance Metrics for Intell. Syst.*, New York, NY, 2009, pp. 268-273.

[102] J. Maes, *et. al.*, "Multilevel capacitated lotsizing complexity and LP-based heuristics," *European J. Operational Research*, vol. 53, no. 2, pp. 131-148, Jul. 1991.

[103] A. Newell, *et. al.*, "Elements of a theory of human problem solving," *Psychological Review*, vol. 65, no. 3 pp. 151-166, May 1958.

[104] D. Garlan and M. Shaw, "An introduction to software architecture," *Advances in Software Eng. and Knowledge Eng.*, vol. 1, pp. 1-40, Jan 1994.

[105] P. Winker and M. Gilli, "Applications of optimization heuristics to estimation and modelling problems," *Computational Stat. & Data Anal.*, vol. 47, no. 2, pp. 211-223, Sept. 2004.

[106] L. B. Hales, *et. al.*, "Adaptive object-oriented optimization software system," U.S. Patent 6 112 126, Aug. 29, 2000.

[107] Surajit Chaudhuri. 1998. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* (PODS '98). ACM, New York, NY, USA, 34-43. DOI=10.1145/275487.275492

[108] M. Wallace, "Practical applications of constraint programming," *Constraints*, vol. 1, no. 1-2, pp. 139-168, Sept. 1996.

[109] L. Cardelli and P. Wegner," On understanding types, data abstraction, and polymorphism," *ACM Computing Surveys*, vol. 17, no. 4, pp. 471-523, Dec. 1985.

[110] J. M. Smith and D. C. P. Smith, "Database abstractions: aggregation and generalization," *ACM Trans. on Database Syst.*, vol. 2, no. 2, pp. 105-133, Jun. 1977.

[111] E. Duesterwald, *et. al.*, "Characterizing and predicting program behavior and its variability," in *Proc. 12th Int. Conf. Parallel Architectures and Compilation Techniques*, 2003, Washington, DC, pp. 220-231.

[112] J. Leon and A. Lori, "Continuous self-evaluation for the self-improvement of software," in *Self-Adaptive Software.* Berlin, Germany: Springer, 2001, pp. 27-39.

[113] R. P. Gabriel and R. Goldman, "Conscientious software," in *Proc. 21st Annu. ACM SIGPLAN Conf. Object-oriented Programming Syst., Languages, and Applicat.,* Portland, OR, 2006, pp. 433-450.

[114] K. Cwalina, and B. Abrams. *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable. Net Libraries*. Boston: Addison-Wesley Professional, 2008.

[115] Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing* (STOC '71). ACM, New York, NY, USA, 151-158. DOI=10.1145/800157.805047

[116] C. M. Macal and M. J. North, "Tutorial on agent-based modeling and simulation," in *Proc. 37th Conf. Winter Simulation,* Orlando, FL, 2005, pp. 2-15.

[117] S. K. Das and D. J. Cook, "Designing and modeling smart environments," in *Proc. 2006 Int. Symp. World of Wireless, Mobile and Multimedia Networks*, Buffalo, NY, 2006, pp. 490-494.

[118] E. Hart and K. Sim. (2013, Aug. 1). *Lifelong Machine Learning Systems & Optimisation* [Online]. Available: http://rollproject.org/lifelong-machine-learning-systems-optimisation/

[119] Z. Tan, *et al.*, "Storing Normalized XML Documents in Normalized Relations," in *5th Int. Conf. Comput. and Inform. Technology*, Shanghai, China, 2005, pp. 123-129.

[120] B. R. Arif. (2011). *On the Footsteps to Generalized Tower of Hanoi Strategy* [Online]. Available FTP:arxiv.org Directory: papers/1112 File: 1112.0631.pdf

[121] M. Szegedy, "In how many steps the k peg version of the towers of Hanoi game can be solved?" in *Proc. 16th Annu. Conf. Theoretical Aspects of Computer Sci.*, Trier, Germany, 1999, pp. 356-361.

[122] Zhengya Sun, Wei Jin, and Jue Wang. 2012. Generic subset ranking using binary classifiers.*Theor. Comput. Sci.* 456 (October 2012), 89-99. DOI=10.1016/j.tcs.2012.05.026 http://dx.doi.org/10.1016/j.tcs.2012.05.026

[123] O. Goldreich, "On Teaching the Basics of Complexity Theory," in *Lecture Notes in Comput. Sci.*, vol. 3895. Heidelberg, Germany: Springer, 2006, pp. 348-374.

[124] E. Horowitz and S. Sahni, "Computing partitions with applications to the knapsack problem," *J. Association for Computing Mach.*, vol. 21, pp. 277–292, 1974.

[125] B. G. Malkiel, "The Efficient Market Hypothesis and Its Critics," *J. Econ. Perspectives*, vol. 17, no. 1, pp. 59-82, Winter 2003.

[126] M. Huskins, *et al.* (2013, Aug. 27). *Enhancing the efficiency and effectiveness of application development* [Online]. Available: http://www.mckinsey.com/insights/business_technology/enhancing_the_efficiency_and_effectiveness_of_application_development

[127] D. M. Kaiser, "Automatic feature extraction for autonomous general game playing agents," in *Proc. 6th Int. Joint Conf. Autonomous Agents and Multiagent Syst.*, Honolulu, HI, 2007, pp. 655-661.

[128] E. Seppanen, *et al.*, "High performance solid state storage under Linux," in *2010 IEEE 26th Symp. Mass Storage Syst. and Technologies*, Incline Village, NV, 2010 pp. 1-12.

[129] X. Yu, "Estimating language models using Hadoop and HBase," M.S. thesis, School of Informatics, Univ. of Edinburgh, Edinburgh, Scotland, 2008.

[130] J. Lin, "Mapreduce is Good Enough? If All You Have is a Hammer, Throw Away Everything That's Not a Nail!," *Big Data*, vol. 1, no. 1, pp. 28-37, Mar. 2013.

[131] W. Fang, *et al.*, "Parallel data mining on graphics processors," Hong Kong Univ. Sci. and Technology, Hong Kong, China, Tech. Rep. HKUST-CS08-07, Oct. 2008.