Algorithm Synthesis through Problem Reformulation

Michael R. Lowry
Stanford Artificial Intelligence Laboratory
Box 3350, Stanford CA 94305
And Kestrel Institute
1801 Page Mill Road, Palo Alto CA 94304

Abstract

AI has been successful in producing expert systems for diagnosis, qualitative simulation, configuration and tutoring-e.g. classification problem solving. It has been less successful in producing expert systems that design artifacts, including computer programs. Deductive synthesis of a design from first principles is combinatorially explosive, yet libraries of design schemas do not have sufficient flexibility for application to novel problems.

This paper proposes that the major factor in applying design knowledge is reformulating a problem in terms of the parameters of generic designs. This paper shows how to represent knowledge of generic designs as *parameterized theories*. This facilitates problem reformulation, making it a well defined search for appropriate parameter instantiations.

The representation of design knowledge with parameterized theories is illustrated with generic local search algorithms. The utility of parameterized theories is shown by deriving the simplex algorithm for linear optimization from specification.

I. Introduction

This paper[1] presents a theory of design and problem solving based upon problem reformulation. The key idea is to reformulate a specific problem into an instantiation of the parameters of a generic problem solving method. Its companion IJCAI87 paper [Lowry, 1987a] describes problem reformulation through abstraction by incorporating important problem constraints. Together they describe the methods that are being implemented in the STRATA automatic programming system.

A parameterized theory is a set of symbols which form a language and a set of axioms which constrain the symbols of the language. Some or all of these symbols are parameters which are instantiated by mapping them to terms in another language. A mapping is valid if the axioms are valid when the terms are substituted for the parameters. Parameterized theories were originally developed as part of a rigorous foundation for abstract data types. They have subsequently been extended to abstract modules and specification languages [Goguen and Burstall, 1985] [Goguen and Meseguer, 1982].

The advantage of using a formal framework is the unification and generalization of previous work, the identification of the key search problems, and a declarative representation. The advantage of using parameterized theories over schemas or skeletal plans is that they can express design knowledge without commitment to any implementation, and can be readily combined, composed, extended, and specialized *without* destructive interference. A calculus for combining theories can be found in [Goguen and Burstall, 1985]. [Lowry, 1987b] gives an example of combining the theory of local search and the theory of GPS to yield selection sort.

The pioneering work of Amarel[Amarel, 1968] 20 years ago showed the potential power of reformulation. Starting about 1980, a number of people began investigating methods for searching the space of logically equivalent problem reformulations. Most methods involved finding problem reformulations targeted to a particular problem solving schema such as Divide and Conquer[Smith, 1985], Heuristic Search[Mostow, 1983], Depth First Backward Chaining[Subramanian, 1986], and [Riddle, 1986]. The work presented in this paper generalizes the work cited above by representing problem solving schemas as parameterized theories. This paper also presents more general domain independent methods of searching for good parameter instantiations, which is the critical bottleneck for this kind of problem reformulation.

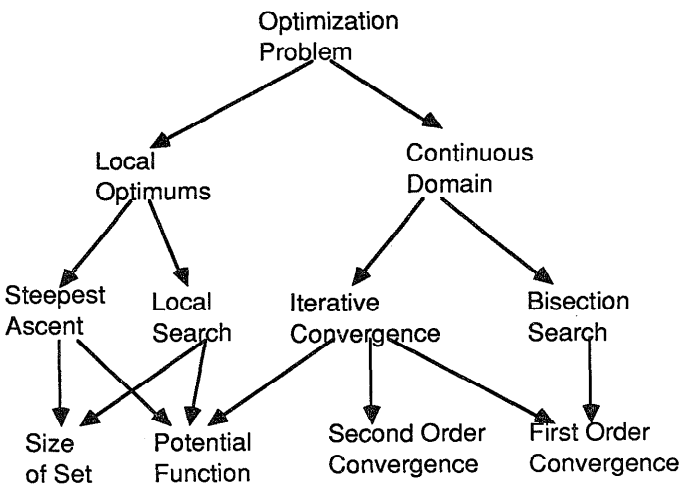
The next section of this paper shows how to represent design knowledge as a refinement hierarchy of parameterized theories. The third section gives a brief overview of the simplex algorithm, and the fourth section shows how to design artifacts by choosing, refining, and instantiating parameterized theories. The derivation of the simplex algorithm is used as an example.

II. Representing Generic Designs as Parameterized Theories

The diagram below illustrates a hierarchical representation space for algorithm design knowledge based upon parameterized theories. The case shown is generic design knowledge for optimization problems, which will later be used

[1]This work was done at Stanford University under DARPA contract N00039-84-C-0211, and at the Kestrel Institute under ONR contract N00014-84-C-0473.

in deriving the simplex algorithm. Each additional level is a refinement of the design knowledge of the previous level. Each level is represented as a parameterized theory which is applied to a particular problem by instantiating the parameters such that the instantiated axioms are provably correct. Each additional level is a specialization (more parameterized axioms) and possibly an extension (more parameters) of the previous level. Thus this hierarchical representation naturally supports top-down refinement of an evolving design without being overly committed to an implementation language.



The parameterized theories which will be used in deriving the simplex algorithm are given below. They correspond to the left hand side of the diagram above, i.e. a generic optimization problem, the applicability conditions of local optimums being global optimums, local search algorithm, and finally a performance guaranteed to be no worse than the size of the domain if no looping occurs. Each theory begins with parameters for sorts, relations, and functions, which are followed by a set of axioms. Each successive layer is represented by the additional parameters and axioms which are added to its parent theory.

Global Input/Output Behavior:Optimization Problems

Domain D States S
Begin, End $:\rightarrow S$
$Value : S \rightarrow D$
$CostRelation : D \times D$
Axioms
$Value(Begin) \in D$
$\forall x \in S\ x = END \leftrightarrow \forall y \in D\ CostRelation(Value(x), y)$
$\forall x, y \in D\ CostRelation(x, y) \vee CostRelation(y, x)$
$\forall x, y, z \in D\ CostRelation(x, y) \wedge CostRelation(y, z) \Rightarrow CostRelation(x, z)$

The global input/output behavior consists of the sort, relation, and function parameters which are used to specify the generic problem. In this example the generic problem is optimization over a domain D. The axioms specify the constraints between the parameters, i.e. the *CostRelation* is a total order upto equivalence - that is all the domain elements are comparable. The *Value* function is a map from a state to an element of the domain. The *Value* function is usually implemented as a program variable, but other implementations are possible. An advantage of parameterized theories is the flexibility of having no a priori commitment to a particular implementation. For an optimization problem, the *Value* in the begin state is some element of the domain, and the *Value* in the end state is an optimal element of the domain. Optimality is determined by *CostRelation*.

Applicability Conditions:
Local Optimums ≡ Global Optimums

$Neighbor : D \times D$
Axioms
$\forall x, y \in D\ TransitiveClosure(Neighbor)(x, y)$
$\forall x \in D\ \{\forall y\ Neighbor(x, y) \Rightarrow CostRelation(x, y)\}$
$\Leftrightarrow \{\forall y \in D\ CostRelation(x, y)\}$

An applicability condition specifies the additional problem structure which is exploited by a general problem solving method. There can be many problem solving methods which exploit the same problem structure, for example both steepest ascent and simple hill climbing exploit the equivalence of local optimums and global optimums. Instantiating the applicability conditions before committing to a particular control and data flow structure is a stepwise refinement reformulation strategy which is readily supported with parameterized theory representations.

The applicability conditions in this example specify *locality* in terms of a *neighbor* parameter. There are two constraints on this *neigborhood* parameter. First, when the *neighbor* relation is viewed as a graph, all domain elements are connected. This ensures that an optimal value is reachable from any initial value. The second constraint is that if a domain element is optimal over its neighborhood, then it is also globally optimal.

Algorithm Structure: Local Search

$Next : S \rightarrow S$
Axioms
$\forall x \in S\ Neighbor(Value(x), Value(Next(x))$
$\forall x \in S\ CostRelation(Value(Next(x)), Value(x))$

The algorithm structure for local search introduces the *Next* parameter, which maps states to states. *Next* specifies large-grained state transitions, usually corresponding to the outer loop of a program. The axioms which are given here specify that the *Value* in the *Next* state is a better neighbor of the current *Value*.

Performance Structure: No Looping

$\text{Size(Domain)} \geq End - Begin$
$\forall x, y \in S\ x \neq y \rightarrow Value(x) \neq Value(y)$

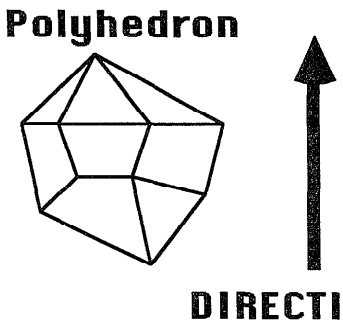
The final layer of the representation of local search design is that of performance. The theory given here is a weak upper bound on the number of state transitions. It states that the number of high-level state transitions from the beginning state to the end state is bounded by

the cardinality of the optimization domain *if there is no looping.*

This section has shown how to represent a general problem solving method as a refinement hierarchy of parameterized theories. Each additional layer introduces more constraints and possibly additional parameters. The significant decision points for problem reformulation are choosing refinements and instantiating parameters. Choosing a refinement is a classification problem-that is, a choice between a small number of alternatives. Often, different refinements lead to equally viable algorithms. Parameter instantiation is a much more difficult search problem, and is the main focus of section 4.

III. Simplex Algorithm Overview

Linear optimization is finding the optimum value of a linear cost function given a set of linear constraints. From the abstract viewpoint of Euclidean geometry, the linear constraints describe a convex polyhedron (possibly unbounded or null), and the cost function describes a direction. The desired output is the point(s) on the polyhedron which is furthest along the direction vector.



The insight of the simplex algorithm is that the output will include a vertex of this polyhedron. The skeleton of the simplex algorithm is local search between adjacent vertices until a local optimum is reached. Because of convexity, a local optimum is guaranteed to be a global optimum.

The standard form of a linear optimization problem is to minimize $c \cdot x$ such that $Ax = b$ and $x_i \geq 0$. The input is a row vector c, a column vector b, and an $m \times n$ matrix A. The output is a row vector x. In the standard form, a vertex is represented by m linearly independent columns of the matrix A, where m is the number of rows and n is the number of columns, n being strictly greater than m. Thus there are m choose n possible representations of vertices(a vertex might have multiple representations). Adjacent vertices share $m-1$ columns. The co-ordinates of a vertex can be explicitly determined by solving the $m \times m$ submatrix of column vectors for b using gaussian elimination. A vertex has m non-zero co-ordinates.

The significant design choices in deriving the simplex algorithm are first the parameterized theory refinements which lead to the choice of a local search algorithm and more importantly the parameter instantiations:

1. Instantiation of the domain of optimization to be just the vertices, which makes the search space finite.
2. Instantiation of the neighbor relation to be vertices which share $m-1$ columns, thus minimizing the search at each step.
3. Specifying a total order on subsets of m columns to avoid looping. The CostRelation only gives a partial order.
4. Instantiating the first phase of the algorithm, which yields a valid starting point for the optimization.

IV. Applying Design Knowledge: Deriving the Simplex Algorithm

This section discusses the use of parameterized theories in designing an algorithm. The full derivation can be found in [Lowry, 1987b], this overview focuses on the methods used in instantiating the parameters (step 3 of the basic method). *Heuristics for instantiating the parameters are themselves represented as parameterized theories and mappings between parameterized theories.*

The basic design method is:

1. Choose a parameterized theory, or refinement.
2. Propagate constraints.
3. Generate problem specific instantiations of free parameters which satisfy the propagated constraints.
4. Iterate until the *Next* parameter is fully constrained, i.e. the algorithm is complete.

Constraints are accumulated on the sequence of state changes, represented by the *Next* parameter. These constraints are then transformed to a set of state transformation rules, which are then compiled by the $REFINE^{TM}$ compiler into lisp code. The input to STRATA is the problem definition, domain knowledge such as theorems of linear algebra, and a library of design knowledge expressed as parameterized theorems. The output of STRATA is the set of constraints on the *Next* parameter.

Input: Problem Definition for Linear Optimization

$Value1(Begin) = (A, b, c)$
$Value2(END) = x^{out}$
$Ax^{out} = b, x_i^{out} \geq 0$
$\forall x \in \{x \mid Ax = b \wedge x_i \geq 0\}\ c \cdot x^{out} \leq c \cdot x$

Partial Output: Constraints on the *Next* parameter

(These constraints are derived by propagating the *neighbor* instantiation to the local search refinement, as explained later.)
$\forall s \in States\ s = END \leftrightarrow$
$\{\forall v \in vertices\ Adjacent(Value2(s), v)$
$\Rightarrow c \cdot Value2(s) \leq c \cdot v\}$
$\forall s \in States\ Adjacent(Value2(s), Value2(Next(s))$
$\forall s \in Statesc \cdot Value2(Next(s)) \leq c \cdot Value2(s)$

The first axiom states that if the current value of x^{out} is locally optimal, then the algorithm should terminate. The following two axioms state that the next value of x^{out} should be a better neighbor of the current value of x^{out}.

The first step in the derivation is to instantiate a generic input/output behavior to the linear optimization problem. The generic optimization problem is partially instantiated with the following representation map:
$D \mapsto \{x \mid Ax = b \wedge x_i \geq 0\}$ ab.[2] *poly*
$CostRelation \mapsto (\lambda(a,b)c \cdot a \geq c \cdot b)$ ab. *LAMBDA*
$Value \mapsto Value2$
$Value(End) \mapsto x^{out}$
$Value(Begin)$ *UNINSTANTIATED*

The instantiated axioms for generic optimization are provably true in the problem domain theory of linear algebra. The uninstantiated parameter is constrained as follows, it is the postcondition for the first phase of the simplex algorithm:
$Value2(Begin) \in poly$

Heuristic knowledge for instantiating the parameters can be encoded in parametric form. This knowledge expresses additional constraints on a parameter and/or specifies how to instantiate a parameter in terms of other parameters. The additional constraints serve to focus the generation of problem specific instantiations of free parameters in step 3. Specifying the instantiation of a parameter in terms of other parameters which are syntactically closer to the domain representation can reduce the 'reformulation distance' that needs to be spanned by equivalence preserving transformations and general purpose theorem proving methods. As an example, one heuristic for instantiating the *DOMAIN* of optimization is to find a predicate which restricts the domain to a subset which includes at least one optimal solution:
$\exists x P(x) \wedge \forall y \in D\ CostRelation(x,y)$
$D' \mapsto \{x \in D \mid P(x)\}$

This heuristic can be invoked as a demon when instantiating the domain parameter of an optimization problem. This heuristic essentially encodes a parameterized proof that restricting the domain of optimization yields a valid algorithm. Given the representation map derived above, and the following theorem found in textbooks on linear programming, this heuristic derives an instantiation for D' which restricts the domain of optimization to vertices, i.e. vectors with only m non-zero co-ordinates:
Thm: $\exists x \in poly\ size\{i \mid x_i \neq 0\} = m$
$\wedge\{\forall y \in poly\ Lambda(x,y)\}$[3]
$D' \mapsto \{x \in poly \mid size\{i \mid x_i \neq 0\} = m\}$ ab. *vertices*.

The next step (step 1 of the basic method) in the derivation is to choose the applicability conditions and instantiate the parameters with appropriate domain functions and relations. A heuristic which is activated when instantiating the *Neighbor* parameter of local search yields small neighborhoods so that the local search of each neighborhood is efficient. This heuristic defines the neighbor relation in terms of the parameterized theory of a distance metric, in particular the minimal distance such that local optimums are global optimums:
$Neighbor(x,y) \mapsto Dist(x,y) < K$
$K \mapsto Minimize(U \mid \forall x \in D\ \{\forall y \in D\ CostRelation(x,y)\}$
$\Leftrightarrow \{\forall y\ Dist(x,y) < U \Rightarrow CostRelation(x,y)\}$
$Dist(x,x) = 0$
$x \neq y \Rightarrow Dist(x,y) > 0$
$Dist(x,y) = Dist(y,x)$
$Dist(x,z) < Dist(x,y) + Dist(y,z)$

To instantiate this heuristic, STRATA first generates a distance metric on D (the domain of optimization), attempts to find a minimal K and if successful instantiates the *Neighbor* parameter. After searching various possibilities, a successful instantiation is found that uses a composite distance function, i.e. $Dist(x,z) \mapsto G(H(x,z))$. G is instantiated first to the primitive function $Setsize$, and some distance metric axioms are back-propagated to constraints on the function H (H has arity setof non-zero coordinates $\times$ setof non-zero co-ordinates $\rightarrow$ setof non-zero co-ordinates):
$\forall x \in vertices H(x,x) = \phi$
$\forall x,y \in vertices\ x \neq y \Rightarrow H(x,y) \neq \phi$
$\forall x,y \in vertices H(x,y) = H(y,x)$

H is instantiated to $SymmetricSetDifference$, the *Dist* parameter is instantiated, and the triangle inequality is verified:
$Dist(x,y) \mapsto SetSize(SymmetricSetDifference(x,y))$

A similar derivation also works in deriving local search algorithms for Minimal Spanning Trees, and approximation algorithms for the Traveling Salesman Problem [Papadimitriou and Steiglitz, 1982]. This suggests that a fruitful line of research is to apply *Explanation Based Generalization*[Winston *et al.*, 1983] to derive new parameterized theories as heuristics for instantiating parameterized designs. For this particular example, the generalization would instantiate the distance metric to $SetSize(SymmetricSetDifference(x,y))$ when the domain of optimization can be formulated as subsets of another set:
IF $D \mapsto \{s \mid s \subset E\}$
THEN
$Dist(x,y) \mapsto SetSize(SymmetricSetDifference(x,y))$

The instantiated *Dist* parameter is then used to instantiate the *Neighbor* relation by finding the minimal K such that local-global optimality is satisfied:
$Neighbor(x,y)$
$\mapsto SetSize(SymmetricSetDifference(x,y) \leq 2$

This instantiation states that 2 vertices are neighbors if they differ by 2 non-zero co-ordinates, i.e. they share $m-1$ column vectors. This instantiation is abbreviated *Adjacent*. When this instantiation is propagated to the local search parameterized theory refinement, it yields the

[2] ab. abbreviates abbreviated

[3] In this derivation it is assumed that there exists a bounded optimal solution.

constraints on the *Next* parameter given at the beginning of this section.

The rest of the derivation uses the same techniques of choosing an incremental parameterized theory to refine the evolving design, propagating constraints, and then instantiating free parameters. The search for parameter instantiations uses the same methods described above, including constraint propagation and heuristics expressed as parameterized theories and representation maps.

This section has shown how design can be factored into a classification problem of choosing a parameterized theory and a reformulation problem of finding appropriate domain terms to instantiate the parameters of generic designs. A major contribution is the demonstration of how domain independent inference techniques, domain independent heuristics, and domain knowledge expressed as theorems can be combined to focus the search for composite terms to instantiate the parameters of parameterized theorems.

V. Summary

This paper has presented a representation for design knowledge based upon *parameterized theories*, which factors the design problem into a classification problem (choose a generic design strategy) and a reformulation problem (reformulate the problem into the parameters of the generic design). The reformulation problem is combinatorially explosive and poorly understood. Previous work has usually used either ad hoc domain specific techniques or brute force generate and test combined with theorem proving for verification. In contrast, parameterized theories naturally support techniques such as constraint propagation, solving for an unknown parameter in terms of known parameters and mutual constraints, and other inference methods. They also provide a convenient matrix for expressing heuristic knowledge for choosing good instantiations of a parameter.

Parameterized theories offer significant advantages over skeletal plans and program schemas. These advantages are due to being able to combine parameterized theories with a flexible and semantically well defined calculus [Burstall and Goguen, 1977]. In my research, this flexibility is used to express design knowledge as a refinement hierarchy of parameterized theories ranging from a generic problem, applicability conditions, algorithm structure, and finally performance calculations. Each additional layer is a specialization (additional axioms) and/or an extension (additional parameters) of the previous parameterized theory.

VI. Acknowledgments

This research benefited from discussions with Professor Thomas Binford, Dr. Yinyu Ye and Irvin Lustig(Stanford–EES, Operations Research), Dr. Joseph Goguen (SRI–Theory of Abstract Data Types), Dr. George Stolfi (Stanford–Computational Geometry), Dr. Douglas Smith and Dr. Cordell Green (Kestrel Institute–knowledge based automatic programming). This paper benefitted from the comments of the referees and the editing help of Raul Duran, Laura Jones, and Patricia Riddle.

References

[Amarel, 1968] Saul Amarel. On representations of problems of reasoning about actions. *Machine Intelligence 3*, 1968.

[Burstall and Goguen, 1977] Rod M. Burstall and Joseph Goguen. Putting theories together to make specifications. In *IJCAI 5*, pages 1045–1058, 1977.

[Goguen and Meseguer, 1982] Joseph Goguen and José Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In *ICALP*, Springer Verlag, 1982.

[Goguen and Burstall, 1985] Joseph A. Goguen and Rod M. Burstall. *Institutions: Abstract Model Theory for Computer Science.* Technical Report CSLI–85–30, CSLI, 1985.

[Lowry, 1987a] Michael R. Lowry. The abstraction/implementation model of problem reformulation. In *IJCAI-87*, August 1987.

[Lowry, 1987b] Michael R. Lowry. *Algorithm Synthesis through Problem Reformulation.* PhD thesis, Stanford University, 1987.

[Mostow, 1983] Jack Mostow. Machine transformation of advice into a heuristic search procedure. In *Machine Learning, An Artificial Intelligence Approach*, chapter 12, Tioga Press, 1983.

[Papadimitriou and Steiglitz, 1982] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, 1982.

[Riddle, 1986] Patricia Riddle. An overview of problem reduction: a shift of representation. In *Workshop on Knowledge Compilation*, pages 91–112, 1986.

[Smith, 1985] Douglas R. Smith. Top–down synthesis of divide–and–conquer algorithms. *Artificial Intelligence*, 27(1), September 1985.

[Subramanian, 1986] Devika Subramanian. Reformulation. In *Workshop on Knowledge Compilation*, pages 119–121, 1986.

[Winston *et al.*, 1983] Patrick Winston, Thomas Binford, Boris Katz, and Michael R. Lowry. Learning physical descriptions from functional definitions, examples, and precedents. In *AAAI-83*, August 1983.