Invariant Logic: A Calculus for Problem Reformulation

Michael R. Lowry
Stanford Robotics Laboratory
Box 3350, Stanford CA 94309
And Kestrel Institute
1801 Page Mill Road, Palo Alto CA 94304
lowry@kestrel.arpa

Abstract

Symmetries abound in nature. Observing symmetries often provides the key to discovering internal structure. In problem solving, observing and reasoning about symmetries is a powerful tool for shifting viewpoints on a problem.

A calculus for reasoning about problem symmetries has been developed, called Invariant Logic. Invariant Logic is partially implemented in STRATA, a system which synthesizes algorithms through problem reformulation.

In STRATA, Invariant Logic is used to reason about generalized problem symmetries for several purposes. The first purpose is as a calculus for generating expressions denoting problem symmetries. The second purpose is problem abstraction - generating abstract problem descriptions which denote models in which the problem symmetries have been collapsed. The third purpose is problem reduction - specializing a problem description by adding constraints in order to realize performance gains.

1 Introduction

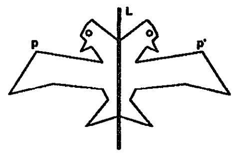
One hundred years ago mathematics was undergoing a revolution. The Kantian dictate that Euclidean Geometry is the only rationally conceivable basis for the physical universe had been debunked. Numerous alternative geometries, each self-consistent, were being discovered, axiomatized, and developed. Felix Klein found a unifying principle for relating and classifying the various geometries - Invariant Theory. The key idea is to classify mathematical structures by the transformations under which they are invariant. Invariant Theory has achieved wide influence in mathematics, physics (including relativity and quantum mechanics), and computer science. The calculus developed here is based upon relatively simple aspects of Invariant Theory.

This paper presents initial work on Invariant Logic- a tool to automate reasoning about symmetries (denoted by groups of transformations) and invariants. In the STRATA system, Invariant Logic is used for problem reformulation, generating abstract data types, and algorithm synthesis. Section 2 overviews the basic concepts of Invariant Logic, illustrated with Euclidean symmetries of geometric figures. Section 3 shows how Invariant Logic can be used to abstract the representation of a simple combinatorial problem. Section 4 demonstrates Invariant Logic applied to generating abstract data types, when given a domain theory and a problem specification. Section 5 explores mathematical aspects related to problem reformulation, including duality and isomorphism between theories. More technical detail can be found in [Lowry, 1988].

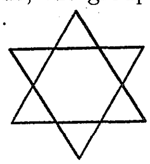
Prior work in problem reformulation and algorithm synthesis has addressed specific aspects of the use of symmetry. Amarel[Amarel, 1968] showed how the symmetry under time reversal of solutions to the missionary and cannibals problem could be used to halve the depth of the search space. Cohen[Cohen, 1977] later generalized this work to the class of state space search problems. Korf [Korf, 1980] gave many interesting examples of the potential use of symmetries in abstracting problem representations, and developed a set of primitive "isomorphic" reformulation rules. The mathematical basis for isomorphism between theories is formalized in section 5 of this paper. Kokar's COPER system[Kokar, 1986] discovers equations for physical laws from experimental data. COPER uses the same mathematical foundation as Invariant Logic, though in a different setting - dimensional analysis. McCartney's Medusa system [McCartney, 1987] uses predefined geometric dual transforms for synthesizing algorithms in computational geometry.

2 Symmetry, Invariance, and Transformations

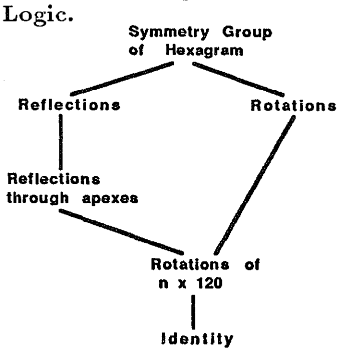
This section describes the underlying concepts of Invariant Logic using geometric figures and shapes.

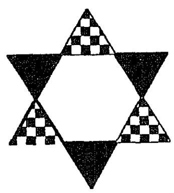


The double headed eagle, crest of the Dukes of Savoy, has bilateral symmetry. It is mapped to itself through reflection about the line l. Reflection about l defines a one-to-one transformation R which leaves the figure invariant. R maps point p to point p', and vice versa. Note that R is its own inverse, applying R twice takes p back to itself: R compose R = identity. R and the identity transformation form a group of transformations. A group of transformations is a set of transformations which include the identity, an inverse for each transformation, and is closed under composition. Closure under composition means that two transformations composed together result in another transformation from the group. In order to be invertible, each transformation must be one-to-one. Formally, the group elements are the transformations, the group operation is composition of transformations.



The hexagram, or Star of David, has both rotational and reflective symmetries. It is invariant under the six rotations about its center (multiples of 60 degrees) and six axis of reflection. In Invariant Logic, this is denoted: $Invariant(hexagram, Rotations\ Join\ Reflections)$. The Join operation takes two groups of transformations, forms the union, and then generates the closure of this union under composition. (An interesting geometric fact is that the closure of the reflective symmetries includes the rotational symmetries.) The Meet operation takes two groups of transformations and returns their intersection. The meet of the rotational symmetries and the 3 reflective symmetries with axis through apexes of the hexagram are the rotations of multiples of 120 degrees. This is because two reflections whose axis form an angle of n x 60 degrees generate a rotation of n x 120 degrees. The group of transformations which leave a geometric figure invariant form a lattice structure with respect to its subgroups, illustrated below for the hexagram. A subgroup is a subset of transformations which are closed under composition and inverses. Notice that as more subgroups are joined together they converge upon the total group of transformations. The search space is commutative and convergent. In later sections of the paper *problem specifications* will be abstracted by discovering *problem symmetries* and incorporating these symmetries into the problem formulation. **Because of the commutative and convergent lattice structure of the subgroups of a group, search control is a minor issue in problem abstraction using Invariant Logic.**





The *orbit* of an object under a transformation group is the equivalence class of objects to which it is transformed. It is an equivalence class because a transformation group includes the identity (reflexivity), inverses (symmetry), and is closed under composition (transitivity). The metaphorical origin of the term *orbit* is illustrated by the orbits of the triangles in a hexagram under rotations of multiples of 120 degrees. The black triangles are mapped to each other, as are the checkered triangles, thereby forming two equivalence classes of triangles in the hexagram. Each equivalence class can be generated by a representative element and the group of transformations.

Symmetry can be used for simplifying representations. With bilateral symmetry only half a figure needs to be given, along with the axis of reflective symmetry. The hexagram, considered as a set of line segments, can be represented by an isoceles triangle (one of the apexes), and the transformation group defined by the six rotations: $Hexagram = Apply(Rotations, triangle)$.

3 Extensional Invariant Reasoning

This section shows how Invariant Logic can be applied to problem abstraction when the semantics are given extensionally, i.e. as an explicit listing of a set. The classic missionary and cannibals problem [Amarel, 1968], is to move 6 people across a river in a 2 man boat without any of them eating each other. Most accounts of this problem begin with the formulation that there are 3 missionaries and 3 cannibals, and a legal intermediate state is one in which the missionaries are never outnumbered and then eaten on either side of the river. However this is already an abstract formulation which incorporates a great deal of relevant information about the legal states. An observer would not see this formulation, instead he would see distributions of specific people on the left and right banks. He would probably begin to notice patterns, especially nearly identical intermediate states which only differed by interchanging specific people.

Just like the triangles of a hexagram described orbits under rotations of multiples of 120 degrees, so do the intermediate states describe orbits under the transformations defined by interchanging specific people. Assume the people are Mike,Max,Cal,Cindy,Cory, and that our observer notes that Mike,Mary,Max are mutually interchangeable, as are Cal,Cindy,Cory. He also notes that the left and right banks can be switched. On this basis he is able to partition the 34 legal states he observes into five orbits. Representative states in the five orbits are given below (the river is represented by !!):

nobody	!! Cal Cindy Cory Mary Max Mike
Cal Cindy	!! Cory Mary Max Mike
Cal Cindy Cory	!! Mary Max Mike
Cal Mary	!! Cindy Cory Max Mike
Cal	!! Cindy Cory Mary Max Mike

These five representative states contain sufficient information to generate the complete set of 34 states *given the transformation group* which arises from the possible interchanges among people and banks. Note that each of these representative states are the smallest representative in their orbits with respect to lexicographic ordering on names. Except for the fourth state, they are all lexicographically ordered. This is the basic idea for problem reduction through transformations - choosing a representation which satisfies additional constraints, such as ordered. **A constraint can be added to a problem description if some representative of each orbit satisfies the additional constraint.** This is especially useful for algorithm synthesis, because a more efficient algorithm can often be synthesized when additional constraints can be assumed in an input-output specification.

While problem reduction chooses a representative for each orbit, problem abstraction generates an abstract description in terms of invariant properties for each orbit.

Invariance under a transformation group is a filter to determine which properties are relevant to an abstract description. The value of an invariant property is shared by every element in an orbit. A set of invariant properties is complete if they uniquely determine an orbit from the abstract values and the transformation group.

As an example, let X be a set, P be the powerset of X, and R be a subset of P. Thus R is a set of subsets of X. Let $AllPi(X)$ denote the group of all possible premutations of the elements in X. A permutation is a one-to-one mapping of the elements in a finite set to themselves. It can be thought of as a reordering of a sequence. If $Invariant(R, AllPi(X))$, this means that the size of the subsets in R is a complete set of invariants. Proof: Let r1 be an element of R, e.g. a subset of X, whose size is n1. Then for any other r2, subset of X with size n1, there is a one-to-one transformation from the elements of r1 to the elements of r2. This transformation is contained in $AllPi(X)$. For any subset of X, si, whose size is not n1, there is no one-to-one transformation from r1 to si. Thus the orbit of r1 contains all subsets of equal size from X and only the subsets of equal size. QED.

The following partial set of rules abstract set-theoretic types in terms of invariant properties. Proofs similar to the one above can be found in [Lowry, 1988]. In the rules, MS is some mathematical structure which is being abstracted. In the missionary and cannibals example, MS is the set of legal intermediate states. MS is invariant under $AllPi(X)$, where X is some set used in MS. R is a subtype of MS. The type declarations are based on the $REFINE^{TM}$ language. $Map(domain, range)$ is the declaration for a partial function from *domain* to *range*. $Set(domain)$ is the declaration for a set with elements from *domain*. $Tuple(domain1...domainN)$ is the declaration for an ordered tuple with successive elements from domain1, domain2, etc. The abstraction of both the subtype and the extension are given.

1. The invariant of a subset of X is its size:
$R : set(X)\ AND\ Invariant(MS, AllPi(X))$
$\Rightarrow R : integer$
The extension is the size of the subset.

2. The proof given above, and embedding rule 1.
$R : set(set(X))\ AND\ Invariant(MS, AllPi(X))$
$\Rightarrow R : set(integer)$
The extension is a set of integers, representing the size of the subsets in R.

3. If the value of a multi-argument function is independent of one of its arguments, then delete the argument.
$R : map(tuple(..X..), range)$
$AND\ Invariant(MS, AllPi(X))$
$\Rightarrow R : map(tuple(....), range)$
Extension: project out the argument whose domain is X.

4. A similar rule for a relation.
$R : set(tuple(..X..))\ AND\ Invariant(MS, AllPi(X))$
$\Rightarrow R : set(tuple(....))$
Extension: project out the argument whose domain is X.

5. A function from domain to range can be transformed to a function from range to subsets of domain - i.e. the domain elements which map to a given range element. The invariant in this rule is the number of domain elements which map to a range element.
$R : map(X, range)\ AND\ Invariant(MS, AllPi(X))$
$\Rightarrow R : map(range, integer)$
Alternatively $\Rightarrow R : bag(range)$
Extension: the range is mapped to the number of elements in the inverseimage.

6. Every function defines an equivalence relation on its domain - the elements which map to the same range element. This partitioning of the domain is the invariant in this rule.
$R : map(Domain, X)\ AND\ Invariant(MS, AllPi(X))$
$\Rightarrow R : partition(Domain)$
Extension: the domain is partitioned.

When X is only a subset of the domain or range Y the following rules apply, where D is the set difference between Y and X. The extensions and subtypes are analogous to the previous rules, but involve tupling to separate X and D.

7. $R : set(Y)\ AND\ Invariant(MS, AllPi(X))$
$\Rightarrow R : tuple(integer, set(D))$

8. $R : map(Y, range)\ AND\ Invariant(MS, AllPi(X))$
$\Rightarrow R : map(range, tuple(integer, set(D)))$

9. $R : map(Domain, Y)$
$AND\ Invariant(MS, AllPi(X)) \Rightarrow$
$R : tuple(partition(Subdomain1), map(Subdomain2, D))$
Subdomain1 is the elements of Domain which map to X, and Subdomain2 is those which don't map to X.

These rules can be applied to obtain an abstract representation for the set of legal intermediate states in the missionary and cannibals problem. These rules are not strictly compositional on subtypes, technical details can be found in [Lowry, 1988]. Each state is a mapping from people to locations, so the set of legal states has the following type:
$set(map(\{Mike, Mary, Max, Cindy, Cal, Cory\}, \{left, right\}))$

The transformation group which leaves the 34 legal states invariant is defined using the $AllPi(X)$ construction:
$AllPi(Mike, Mary, Max)\ Join\ AllPi(Cal, Cory, Cindy)$
$Join\ AllPi(left, right)$. This transformation group is composed of three subgroups which are joined together. These subgroups will be used in successive rules to abstract the representation.

Rule 8 uses $AllPi(Mike, Mary, Max)$ to obtain the abstracted type:
$set(map(\{left, right\}, tuple(integer, set(Cal, Cory, Cindy))))$

Rule 1 then uses $AllPi(Cal, Cory, Cindy)$ to obtain:
$set(map(\{left, right\}, tuple(integer, integer)))$

Finally Rule 5 uses $AllPi(left, right)$ to obtain:
$set(bag(tuple(integer, integer)))$

The extension for this abstract type is given below, with the corresponding representative state from each orbit.

```
bag((0,0),(3,3)) nobody !! Cal Cindy Cory Mary Max Mike
bag((0,2),(3,1)) Cal Cindy !! Cory Mary Max Mike
bag((0,3),(3,0)) Cal Cindy Cory  !!  Mary Max Mike
bag((1,1),(2,2)) Cal Mary  !! Cindy Cory Max Mike
bag((0,1),(3,2)) Cal  !! Cindy Cory Mary Max Mike
```

4 Intensional Invariant Reasoning

This section describes how STRATA uses Invariant Logic for abstracting a problem when the semantics are given intensionally, i.e. as a theory. First the rules for computing symmetries for composite relations are given. Then a simple problem is introduced, and it is shown how the Knuth-Bendix completion algorithm can be used to calculate additional problem symmetries. A special type of symmetry - *congruences* - are described. The application of Invariant Logic to the abstraction process is shown, and the alternative of specialization through problem reduction is described.

The following rules of invariant logic provide a calculus for determining the invariants of a composite relation based on the invariants of its subparts. The primary observation for boolean operations on relations is that the result is invariant under the intersection (meet) of transformation groups for the separate arguments. This supports "greatest common divisor" reasoning on composite relations. In the rules which follow, R is a relation i.e. type tuple(domain1,domain2....domainN), and TG is a transformation group over the tuples. These rules are slightly simplified versions of ones in [Lowry, 1988], which address some technical issues concerning whether the domains are distinct.

If a relation is invariant, then so is its complement:
$Invariant(R, TG) \equiv Invariant(\neg R, TG)$

Boolean operations such as union and intersection preserve invariance with respect to the meet of transformation groups (R1,R2 are sets of tuples of the same type):
$Invariant(R1, TG1)\ AND\ Invariant(R2, TG2)$
$\Rightarrow Invariant(R1\ boolop\ R2, (TG1\ meet\ TG2))$

The cartesian product over relations preserves invariance under the direct product of transformation groups.
$Invariant(R1, TG1)\ AND\ Invariant(R2, TG2)$
$\Rightarrow Invariant(R1 \otimes R2, TG1 \otimes TG2)$

If a relation is invariant under a transformation group, then it is invariant under any of its subgroups:
$Invariant(R, TG)\ AND\ Subgroup(TG1, TG)$
$\Rightarrow Invariant(R, TG1)$

Consider the problem of common-members: given two lists as input, output a list whose elements are the common members of the two lists. Abstractly, this problem is simply set-intersection. The problem can be specialized by problem reduction to assume the two input lists are ordered, an efficient algorithm is to march down the two ordered lists in tandem. The reduction is achieved by sorting the two input lists, which is a transformation which leaves the problem invariant.

STRATA generates abstract problem descriptions from a concrete problem description and a domain theory. The conceptual foundation is partially described in [Lowry, 1987], this section discusses interesting aspects of the implementation not covered earlier. For common-member, the domain theory is that of lists; the following equational theory is given to STRATA, where variables are denoted by & . Equational theories are often used for specifying abstract data types. The equations can be turned into a logic program by making one side of each equation into the left hand side of a rewrite rule, and the other side into the right hand side. Equality between ground terms can be decided by rewriting them to normal form with the derived rewrite rules.

```
(append nil &l3) = &l3
(append(cons &a &l)&l3)=(cons &a (append &l &l3))
(member &x nil) = false
(member &x (cons &y nil)) = (EQ &x &y)
(member &x (append &L1 &l2))
  = (member &x &L1) OR (member &x &L2)
;; equations defining semantics of AND, OR, IFF
(common-member &L1 &L2 &L3) =
(member &x &L3) IFF
(member &x &L1) AND (member &x &L2)
```

The first step is to generate the problem symmetries. The rule for boolean composition of relations deduces that the symmetries of member are a subset of the symmetries of common-member. The symmetries of member are not known, and the compositional calculus does not apply (the rules above don't handle recursive definitions). The homomorphism method described in [Lowry, 1987] finds the symmetries of member; it is implemented by applying the Knuth-Bendix completion algorithm. The basic idea in setting up the Knuth-Bendix algorithm is to make *two* copies of the problem name, and linking them. One copy has a heavier weight than any other symbol (e.g. member). The other copy has a lighter weight than any other symbol (e.g. member'). Intuitively, the K-B algorithm percolates the problem definition through the domain theory to the lighter weight problem name, generating problem symmetries.

```
**Derived from K-B when member defined with append**
(member' &x (append &y &z)) =
     (member' &x (append &z &y))
(member' &x (append &y &y)) =
     (member' &x &y)
**Derived from K-B when member defined with cons**
(member' &x (cons &a (cons &b &L))) =
   (member' &x (cons &b (cons &a &L)))
(member' &x (cons &a (cons &a&L))) =
   (member' &x (cons &a &L))
```

These derived theorems denote the symmetry of lists under transformations of reordering and deleting repeated elements. *Structural Induction* is used to verify that new equations can be extracted from these theorems. An equation represents a special kind of problem symmetry (a congruence), which, when substituted into itself still denotes a problem symmetry. As part of the structural induction proof for the commutativity of append, STRATA generates and proves the following theorem:

```
(member' &x (append (append &x &y) &z)) =
(member' &x (append (append &y &x) &z))
```

After proving structural induction, STRATA extracts the following axioms and adds them to the theory of lists, thereby deriving the abstract data type for sets. The added axioms for append make it commutative and idempotent, thus the semantics are set-union, and append **becomes set-union** (names don't matter for the denotation of a theory):

```
Invariant(common-members,
              (append' &y &z) = (append' &z &y))
Justifies add-axiom((append' &y &z)=(append' &z &y))
Invariant(common-members,(append' &x &x) = &x)
  Justifies add-axiom((append' &x &x) = &x)
```

An alternative to abstraction is problem reduction - adding constraints which can be achieved with the transformations which leave the problem invariant. In contrast to abstraction, there are many possible problem reductions because there are many possible representative elements in each orbit. One source of constraints are derived preconditions for operators. In this example, if the lists are ordered an inexpensive necessary condition on membership is that the first element of the list is less than or equal to the element being tested for membership. This gives rise to the derived precondition that the input lists are ordered, which can be achieved by repeatedly switching adjacent list elements which are out of order (bubble sort). This switching transformation is denoted by one of the derived equations for cons:

`(cons &a(cons &b &L))=(cons &b (cons &a &L))`

See [Lowry, 1988] for the application of problem reduction to synthesizing Karmarkar's linear optimization algorithm.

5 Duality and Isomorphic Theories

Duality can be expressed as a symmetry among the *symbols* of a theory which leave the true sentences invariant. A transformation from symbols to symbols is a representation map, designated $RMap$. Duality has bilateral symmetry, an example is boolean algebra (Not is self dual):

$And \longleftrightarrow Or$

$true \longleftrightarrow false$

$Not \longleftrightarrow Not$

This representation map transforms true sentences to true sentences:

$(x\ And\ true) = x \longleftrightarrow (x\ Or\ false) = x$

Duality of a theory is easy to verify - simply transform the axioms with the representation map, and prove the transformed axioms using the original theory: $Dual(Theory, RMap) \equiv Axioms \vdash RMap(Axioms)$ Because proofs are invariant under renaming of symbols, we obtain for free the dual proofs by applying the representation map, which is its own inverse:

$RMap(Axioms) \vdash Rmap(Rmap(Axioms)) = Axioms$

Duality is often exploited in algorithms. Mini-max search and alpha-beta pruning use duality to efficiently search the space of look ahead moves in a competitive game. Linear optimization problems, particularly the special class of network flow problems, can be efficiently solved by primal-dual algorithms. In geometric algorithms, the duality between lines and points in 2-D projective geometry can be used to expand theuses of subroutines. For example, collinearity of points can be mapped to intersection of lines.

In duality, the representation map is from symbol to symbol. Isomorphic theories are defined by generalizing the representation map from symbols to *terms*. A representation map from symbols to terms is not invertible, so the definition is more complex. Theory A and theory B are isomorphic iff there exists representation maps R1 from A to B and R2 from B to A which satisfy:

1. $Axioms(B) \vdash R1(Axioms(A))$
2. $Axioms(A) \vdash R2(Axioms(B))$
3. $R1 \circ R2 = Identity(A)$
4. $R2 \circ R1 = Identity(B)$

Boolean algebra defined with primitives $Nand, true, false$ is isomorphic to boolean algebra defined with primitives $And, Or, Not, true, false$. The representation maps $R1, R2$ are given below:

$Or(x,y) \rightarrow Nand(Nand(x), Nand(y))$

$And(x,y) \rightarrow Nand(Nand(x,y))$

$Not(x) \rightarrow Nand(x)$

$Not(And(x,y)) \leftarrow Nand(x,y)$

The last two conditions on isomorphism between theories is that the composition of representation maps yields the identity. This entails that an equivalence be proven, for $Nand$ the composition of representation maps yield:

$Nand(x,y) \rightarrow Not(And(x,y))$

$\rightarrow Nand(Nand(Nand(x,y)))$ The following equivalence has to be proved in order to show that $R1 \circ R2$ is the identity:

$Nand(x,y) = Nand(Nand(Nand(x,y)))$.

Similar equivalences are needed for And, Or, Not.

An alternative definition of isomorphism between theories is that each can be conservatively extended with defined relations and functions to include the other. Isomorphism between theories is one way to define reformulation.

6 Acknowledgements

This paper benefitted from the technical and editing help provided by Tom Binford, Leonid Frants, Joseph Goguen, Walter Hill, Douglas Smith, George Stolfi, and Yinyu Ye. Special thanks go to the reviewers, who made substantial comments on clarifying the paper. This work was supported in part by DARPA sub-contract AIADS-S10935-1 and ONR contract N00014-87-K-0550. The views and conclusions in this paper are solely those of the author.

References

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

[Cohen, 1977] Brian Cohen. The mechanical discovery of certain problem properties. *Artificial Intelligence*, 8, 1977.

[Kokar, 1986] Mieczslaw M. Kokar. Determining arguments of invariant functional descriptions. *Machine Learning*, 1, 1986.

[Korf, 1980] Richard E. Korf. Towards a model of representation change. *Artificial Intelligence*, 14(1), April 1980.

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

[Lowry, 1988] Michael R. Lowry. *Algorithm Synthesis through Problem Reformulation.* PhD thesis, Stanford University, 1988.

[McCartney, 1987] Robert D. McCartney. Synthesizing algorithms with performance constraints. In *AAAI-87*, 1987.