

A Declarative Theory for Rational Controllers

Wolf Kohn

Boeing Computer Services
P.O. Box 24346
Seattle, WA. 98124-0346

Abstract

This paper presents a computationally effective representation theory of a class of digital controllers herein referred to as Rational Controllers. The theory, which is expressed in terms of first-order predicate logic with some meta-extensions, characterizes the dynamic behavior of an element of the class using equations and inequations that declare the structure of the controller in an algebraic variety V whose algebras satisfy the Central Factorization Principle. The Central Factorization Principle states that an element of an algebraic variety V is either primitive or else can be expressed in finitely many different ways in terms of operations of the algebra on primitive elements. Important elements of V are the algebra of rational sets over the ring of real numbers, the algebra of rational trees, the algebra of rational functions on a module, the algebra of modular lattices and direct products and limits of these algebras.

I Introduction

The formal definition of a control design problem involves the specification of 4 data items: plant dynamics, goal dynamics, control requirements and design strategy. The central objective of this paper is to specify a uniform representation of these items and of the resulting controller that is computationally effective.

A standard controller design process is summarized by the block diagram of Figure 1.

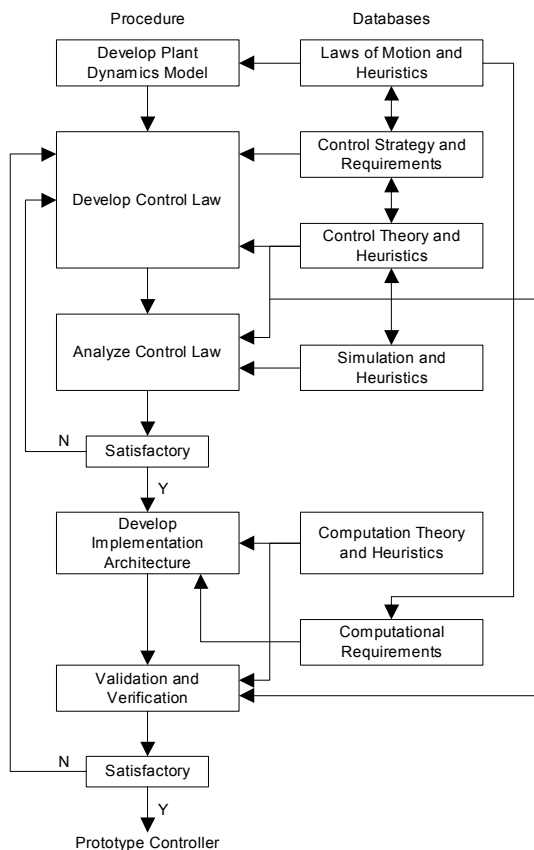


Figure 1: Common Controller Design Procedure

In this process, the designer consults a data base¹ of first principles and heuristics about the system to be controlled and develops a model of the system that captures those features of its dynamics that are relevant to the design. The outcome of this effort is a model of the system referred to herein as the plant dynamics. This model is used in conjunction with the specified control requirements to develop a control law.

The control law is an algorithm that takes sensor and goal command data and generates actuator commands. This algorithm is repeatedly executed according to real-time requirements.

Once a candidate control law has been determined, the designer evaluates it using analytic and simulation-based tools and procedures to verify compliance with the requirements and to evaluate performance.

After the analysis step is completed, the next step in the design is to design an implementation architecture and then to validate and verify its performance against embedded computational principles and computational requirements. Note the two iteration steps in Figure 1.

Although in practice the design scheme suggested by the flow chart of Figure 1 is not followed exactly, the formalism is useful for contrasting it with the one followed in this study, which is shown in Figure 2.

The basic idea behind the controller design procedure considered in this study is to encode the data bases on the right hand side of Figure 1 using the declarative expressiveness of logic programming and then use an inference procedure to emulate the procedural input/output maps of each of the processes in the left hand side in Figure 1. The resulting design process then becomes the prototype controller.

Moreover, at the conceptual level, it is proposed to eliminate the task of deriving the control law algorithm and replace it with an inference mechanism which generates actuator commands as a function of sensory data and goal commands at the appropriate update times. The concept is illustrated with the Prolog pseudo code of Figure 3.

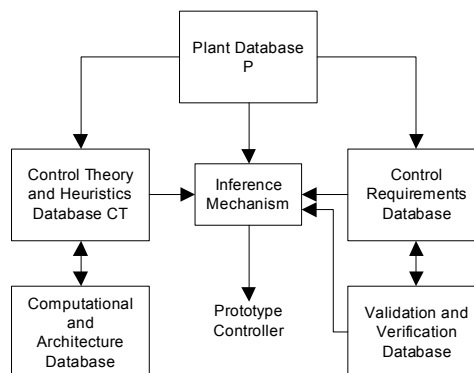


Figure 2: Declarative Design Procedure

```

controller_output(Goal, Sensor, Actuator) :-
    assert(Goal) and
    assert(sensor) and
    data base(Actuator).
  
```

Figure 3: Prolog Pseudo Code for Declarative Controller

¹Here data base does not necessarily mean computer data base.

The suggested design and controller implementation approach as it stands is not practical because even for simple systems the number of logic clauses required to characterize it and the corresponding control requirements is formidable.

Nevertheless, an important empirical observation is that in determining a particular control law, the designer only accesses a very small percentage of the total potential information in the data bases.

Moreover, in the resulting controller, to compute the current actuator command, only a small percentage of the clauses defining controller dynamics are required. This is referred to as the local condition and plays an important part in making the intended representation computationally effective [1].

Although the intent is to make the controller declarative, by definition the controller C is a mapping of the cartesian product goal and sensor spaces into the cartesian product of the actuator and evaluation spaces (see Figure 4(a)), that is, the semantics of the controller is a procedure. Therefore it is necessary to establish a mechanism for translating chains of logic clauses into procedures. This is accomplished by realizing the controller mapping C as a topological locally finite automaton, as shown in Figure 4(b) [2]. Then the clauses representing the control requirements can be stated to define the state transition function δ and the output function β of the controller automaton.

In synthesis, the objectives of this paper are to show how the procedural character of declarative controllers can be effectively represented with generalized automata for a wide class of problems.

The rest of the paper is organized as follows. In Section 2, an overview of the rational variety of algebras is presented. This variety is the fundamental building block of the class of controllers that are under consideration in the study. In Section 3, the structure of the controller is overviewed. In Section 4, a declarative implementation of the controller is proposed and a translator from declarative to procedural execution is presented. Finally in Section 5, some conclusions and future directions are discussed.

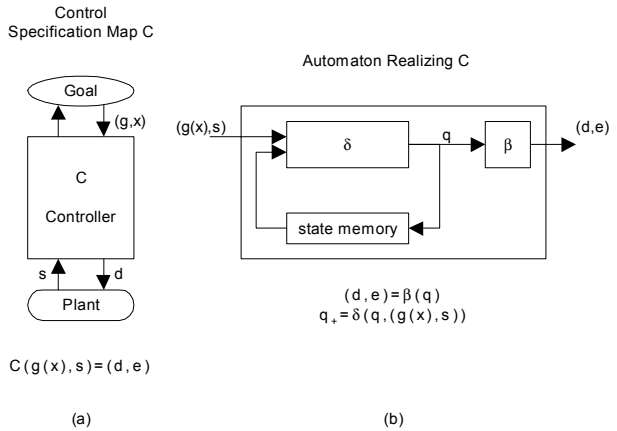


Figure 4: Conversion of I/O Map into Automaton

II Rational Variety

A variety [3] V of algebras is a class of similar algebras which is closed under construction of subalgebras, construction of homomorphic images of algebras in the class, and construction of direct products and direct limits of algebras in the class.

A rational algebra $\langle A, F \rangle$ is an algebra in which the basic operation set F contains the following operations:

- 1) An (infix) additive binary operation denoted by t . This operation has a unit (0) and is associative but not commutative.
- 2) A multiplicative binary associative unit operation with unit 1 which also satisfies the standard distributive operation. This operation is denoted by \cdot .
- 3) A unary operation, denoted by $*$, referred to in the literature as the recursion operator.
- 4) A unary operation, called the annihilation operation, denoted by ϕ . ϕ is defined as follows:

$$\phi(a) = 1 \quad \text{for all } a \in A$$

- 5) A finite but arbitrary set Φ of unary operations, referred to herein as the custom operators:

$$\Phi = \{fr \mid r \in R, fr : X \rightarrow X, X \subseteq A\} \quad (1)$$

with R an index set.

In general, the elements of Φ are partial functions on A .

The star operation is short hand for

$$a^* = 1 + a + a \cdot a + \dots \quad a \in A \quad (2)$$

For simplicity of notation, (2) will be written as

$$a^* = \sum a^i$$

A term operation of the algebras is a function on A constructed of basic operations in f by composition.

The algebra $\langle A, F \rangle$ is rational if every term operation w of the form

$$w = (a)^* \quad (3)$$

where a is a term operation, has at the most a finite number of non-zero terms in the summation.

Examples of algebras that exhibit this characteristic are algebras of rational sets over complete semi-rings [2], the algebra of rational trees [4] and the algebra of finite state automata. A summary of the non-custom operations in two of these algebras is shown in Figure 5.

| | Tree algebra | Automata algebra |
|-----------|--------------------------|---------------------|
| A | Set of trees | Set of automata |
| \cdot_a | replacement | series connection |
| $+_1$ | Colmerauer's constructor | parallel connection |
| $*_a$ | recursion | cascade |
| $-_a$ | cut | anihilation |

Figure 5: Non-custom Operations in Two Rational Algebras

An extensive list of the axioms for these algebras can be found in [5]. For the purposes of this paper it would take too much space to present them.

An important remark is in order: the custom operations make the class of algebras that describe the

structure of rational controllers to be tailored to a particular problem. The custom operations are not constrained a priori to satisfy any axioms besides the finiteness axiom. For each particular controller design problem, specific axioms, represented as logic clauses, will characterize a finite subset of the custom operations. These operations then carry the structural information associated with the problem under consideration.

The axioms of an algebra $\langle A, F \rangle$ can always be represented by a set of identities of the form

$$p_i(X_1, \dots, X_n) = q_i(X_1, \dots, X_n) \quad i = 1, \dots, 4 \quad (4)$$

where p_i, q_i are term operations and X_j ($j = 1, \dots, n$) is a variable taking values in A .

Equations of the form of (4) are interpreted as follows if $X_j = A_j \in A$, $j = 1, \dots, n$, is an assignment, $p_i(a_1, \dots, a_n)$ exists $q_i(a_1, \dots, a_n)$ exists then

$$p_i(a_1, \dots, a_n) = q_i(a_1, \dots, a_n)$$

The variety V of rational algebras is completely characterized by a finite set of identities carrying information about the algebraic structure. In every control design problem, additional equations and inequations are required for incorporating the control requirements, properties of the plant dynamics and of the goal dynamics. In the rational variety V these equations and inequations are either Kleene equations [?] or variations of them. The structure of these equations is shown in the 10 forms listed in Figure 6. Equation 1 is of the Kleene form. In it, $X = (X_1, \dots, X_n)^T$ is a vector of indeterminates. $E_1 = \{E_1^{ij}\}$ is a matrix of rational entries that do not include any of the indeterminates X_i . T is a vector $(T_1, \dots, T_n)^T$ of rational entries.

Equations 2 and 3 are extensions to the basic Kleene form. In equation 4, $X = \{X_{ij}\}$ is a matrix x of indeterminates, E_1, E_2 are matrices of rational entries and T is a vector of rational entries of the appropriate dimensions. This equation is known as a structural Lyapunov equation. In rational controllers, equations of this type code for stability requirements.

For the purposes of this study, the importance of representing the controller requirements with sets of simultaneous equations of the form shown in Figure 6

- | |
|-------------------------------------------|
| 1. $X = E_1X + T$ |
| 2. $X = XE_1 + T$ |
| 3. $X = E_1X + XE_2 + T$ |
| 4. $X = E_1XE_2 + T$ |
| 5. $X = (F_0 + \sum F_iY_i)X + T$ |
| 6. $X = X(F_0 + \sum F_iY_i) + T$ |
| 7. $X = (F_0 + \sum Y_iF_i)X + T$ |
| 8. $X = X(F_0 + \sum Y_iF_i) + T$ |
| 9. $X = (F_0 + \sum Y_iF_{i,j}Y_j)X + T$ |
| 10. $X = X(F_0 + \sum Y_iF_{i,j}Y_j) + T$ |

Figure 6: Requirements Equations

is that there exists an effective procedure for building an algorithm that computes solutions of these equations [7]. This algorithm is a generalized automaton.

Typically, indeterminates will code for actuator commands and evaluation data (see Figure 4).

The automaton obtained from the requirement equations is the controller that satisfies them. Some characteristics of this automaton are discussed in the next section.

III Controller as an Automaton

A controller design is driven by three data items: the goal class, the control specifications, and the robot dynamics. These items are briefly described next.

The goal class represents the set of tasks the robot is required to accomplish. In this study, the goal class G is characterized by two objects: a topological space v and the set of primitive goal actions Ω_G . Symbolically,

$$G = \langle v, \Omega_G \rangle \quad (5)$$

In (5), Ω_G is a set of functions.

$$g_r : X \rightarrow X \quad r \in F_G$$

where $X \subset v$ is a subspace of v , called the goal state set, and F_G is a countable index set. The primitive goal action functions are continuous in the topology of v . A goal task g is in the class G if there exists a *finite* subset of primitive actions g_{r_1}, \dots, g_{r_n} in Ω_G such that the goal task is represented by the functional composition

$$g : X \rightarrow X \\ g = g_{r_1} \cdots g_{r_n} \quad r_i \in F \quad (6)$$

and g is continuous in the topology of G .

Given a goal task g in G and a state $x \in X$, the goal next state is the state $y \in X$ such that

$$y = g(x) \quad (7)$$

Thus the set of primitive goal action functions are a basis for a continuous locally finite semigroup. They represent the elementary maneuvers the robot can accomplish. Composite goals are generated by functional composition of primitive goal actions as indicated in (6).

The goal state is a suitably abstracted representation of the status of the robot. For example, in a two-dimensional mobile robot the goal state set represents the set of symbolic locations and the primitive goal action functions represent the connectivity graph among those locations.

Typically, if x is the current goal state and y is a desired state, a feasible goal is an ordered sequence of primitive goal actions $g_{r_1}, g_{r_2}, \dots, g_{r_n}$ such that

$$y = g_{r_n}(\dots g_{r_2}(g_{r_1}(x)) \dots) \quad (8)$$

It is assumed in this study that during operation of the robot the particular sequence for accomplishing a goal is an input to the system.

The dynamics A is characterized by 4 data items: a topological vector space D , the set of primitive robot command transitions Ω_A and the sensor function s , and the sensor space S

$$A = \langle D, \Omega_A, s, S \rangle \quad (9)$$

where Ω_A is a set of functions

$$d_l : W \rightarrow W \quad l \in F_A$$

with W a subspace of D and F_A is an index set.

The functions in Ω_A are continuous in the topology of D .

The space W is referred to as the state of the system. A state transition from state x to state y is feasible if there exists a finite subset of functions in Ω_d , d_{l_1}, \dots, d_{l_m} , $\{l_i\} \subset F_A$ such that

$$y = d_{l_m}(\dots d_{l_1}(x)\dots) \quad (10)$$

The Ω_A under function composition forms a locally finite continuous semigroup.

The sensor function s maps the space W into a topological vector space S representing the signals generated by the sensors.

That is,

$$s : W \rightarrow S$$

with s continuous with respect to the topologies in W and S .

Finally, the control specifications are represented by a set of clauses in a first-order logic theory [8] characterizing the desired behavior of the robot. These clauses include basic performance specifications such as stability and robustness and task-dependent specifications such as speed of response, terminal accuracy and trajectory shape constraints.

The control specifications also include an inference mechanism called a realization rule which transforms a set of rules into an executable procedure whose dynamics is represented by a topological locally finite automaton. This mechanism is described in detail in [9].

The central idea is that the control specifications are faithfully represented by a topological locally finite automaton which in turn represents the robot controller dynamics.

Conceptually the controller can be represented by a continuous map C of the form

$$C : G \times S \rightarrow D \times E \quad (11)$$

where E is a topological subspace of v called the goal evaluation space. In general E will be a homeomorphic image of S . Each element of E represents the status of the robot in goal space as determined by the robot sensor signals.

Specifically, if g is the current goal task, x is the state of the goal, s is the current sensor signals, d is the current command transition, and e is the homeomorphic image of s in E ,

$$C(g(x), s) = (d, e) \quad (12)$$

This is illustrated in figure 4(a).

The map connecting the goal G and plant dynamics D is the semantic operator [8] of the set of clauses representing the control specifications.

A topological locally finite automaton is a tuple $\langle \Sigma, Q, \delta, B, Y \rangle$ where Σ is the input space, Q is the state space, δ is the state transition function, B is the output function, and Y is the output space. Σ , Q , and Y are topological spaces. The state transition function

$$\delta : Q \times \Sigma \rightarrow Q$$

is continuous in the induced topology on $Q \times \Sigma$ and satisfies the following local conditions. For every q in Q , the map

$$\delta(q, \cdot) : \Sigma \rightarrow Q \quad (13)$$

has a range that is finite or at the most countable, and for all σ_1, σ_2 in Σ ,

$$\delta(q, \sigma_1 \sigma_2) = \delta(\delta(q, \sigma_1), \sigma_2) \quad (14)$$

the space Σ is provided with a binary operation \cdot that is continuous in the topology of Σ and such that the pair (Σ, \cdot) form as a semigroup.

The output function B is a continuous map

$$B : Q \rightarrow Y$$

A locally finite topological automaton $\langle \Sigma, Q, \delta, B, Y \rangle$ is a realization for a controller map C of the form of (8) if the following conditions hold:

$$\begin{aligned} \sigma &= G \times S \\ Y &= D \times E \end{aligned} \quad (15)$$

and if for $g \in G$, $x \in X$, $s \in S$ such that $C(g(x), s)$ is defined, then

$$C(g(x), s) = B(\delta(q, (g(x), s))) \quad (16)$$

for some $q \in Q$.

The controller behavior is fully characterized by the locally finite automaton realizing its input output map. This automaton operates by inferring the state transition and output from the clauses characterizing the control performance specification.

That is, if q is its current state, the goal state is x , and g is the task goal, then the inference procedure alluded to above computes q_+ and (d, e)

$$\begin{aligned} q_+ &= \delta(q, (g(x), s)) \\ (d, e) &= B(q) \end{aligned} \quad (17)$$

This is illustrated in figure 4(b). In general, even for a simple robot and elementary tasks, the number of clauses required to completely characterize the automaton associated with the controller is very large (~ 10000 clauses for a pick-and-place 3-axis manipulator). However for a given state transition $\delta(q, (g(x), s))$ only a small percentage of all the clauses are involved in determining the new state. (That is the essence of the local finiteness of δ .)

IV Declarative Implementation of Generalized Automata

In this section a declarative implementation of the automaton controller is presented. The objective is to characterize its state dynamics (δ) and output (β) with logic clauses and to show how this encoding and an inference mechanism called *eval*, written in Prolog, can be used to run the controller.

First, note that the state transition function δ can be fully represented by a family of unary operations on the state as follows: for each $g \in G$, $s \in S$, $x \in X$ define

$$\Phi_{g,s,x} : Q \rightarrow Q$$

by

$$\Phi_{g,s,x}(q) = \delta(q, (g(x), s)) \quad (18)$$

The set Ω of the form of (18) is called the type of the controller. This set forms a semigroup under composition and it has a basis: the subset of functions $\Phi_{g_i,s,x}$ where g_i is a primitive task and for all $s \in S$, $x \in X$.

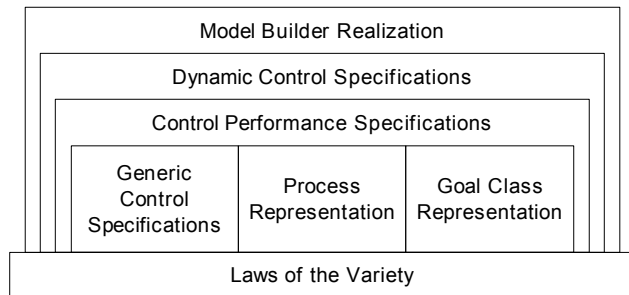


Figure 8: Clause Universe of a Rational Controller Specification

The *eval* procedure is shown in a Quintus Prolog implementation in Figure 7. It works as follows: given a symbolic expression for a type it recursively determines and chains the clauses associated with its primitives and then resolves the chain to find the value of the evaluation (or its non-existence). Figure 7 also shows an example of the clauses (2) for a particular function in the type called *polmci*.

For effective execution at run time it is convenient to organize the data base of clauses in a hierarchical fashion as illustrated in Figure 8.

An implementation of the automata controller can be parallelized even further by partitioning the clauses in the hierarchy according to functionality. One such functional partitioning, currently under investigation, is shown in Figure 9. This architecture was patterned after the model-following paradigm [10].

V Conclusions

A representation theory of a class of controllers, known as rational controllers, has been presented. The theory allows for the representation of the controller dynamics using logic clauses. This declarative representation allows for the incorporation of the design process as a part of the controller.

Currently the theory is being customized for the development of a controller for NASA's Flight Tele-robotic servicer FTS.

```

/* EVALUATION OF ELEMENTS OF THE TYPE */
/* Clause for evaluation of any function in the type */
/* The function is of the form E = op(args),
   the evaluation instance is given in F */
/* args is either a value or of the form op(args1). */
eval(E,F) :- functor(E,Op,N),
              arg_exp(E,N,F_list),
              eval(F_list,New_list),
              op_pred(Op,Pred,_),
              1,
              conc([Fred],New_list,F1),
              conc(F1,[F],F2),
              Y .. F2,
              call(Y).

/* Example of an element of op_pred */
op_pred(polmc1,pol_mcd1,_). /* maximum common divisor of two pols.
                           relative to and order estimate */

/* Corresponding clauses */
pol_mcd(R,pol([mon(_,0)],_),R) :- !.
pol_mcd1(A,B,R) :- pol_div(A,B,_,R2),
                  pol_mcd1(B,R2,R).

```

Figure 7: Eval Mechanism

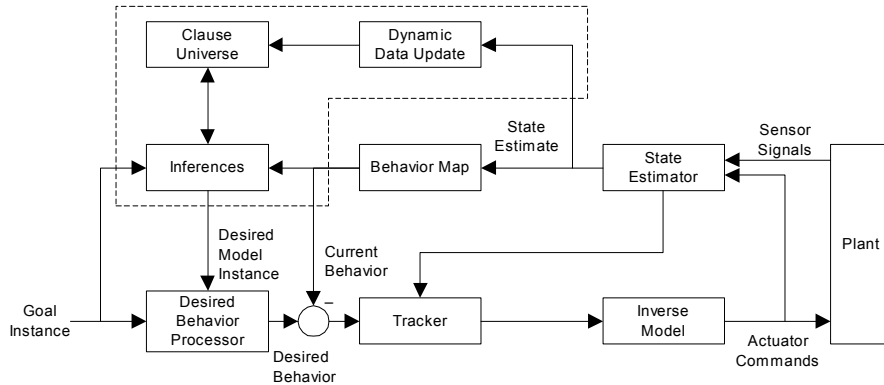


Figure 9: Architecture of a Rational Controller

References

- [1] Gratzner, G., "Universal Algebra," Second Edition, Springer Verlag, N.Y., 1979
- [2] Eilenberg, S., "Automata Languages and Machines," Volume A, Academic Press, N.Y., 1974
- [3] McKenzie, R. G., McNulty, G., Taylor, W., "Algebras Lattices, Varieties," Volume 1, Wadsworth & Brooks/Cole, Monterrey, CA, 1987
- [4] Colmerauer, A., "Prolog and Infinite Trees in Logic Programming," K. L. Clark, S. A. Tarnlund Eds., Apic Studios in D.P. No. 16, 1982, AP, London
- [5] I&RD Report BEC488 Intelligent Hierarchical Control, Seattle, December 1987
- [6] Guinsborg, A., "Automata Theory," Academic Press, 1972, N.Y.
- [7] Kuich, A. Salomaa, "Semirings, Automata Languages," Springer Verlag, 1986, N.Y.
- [8] Loyd, R., "Introduction to Logic Programming Theory," 2nd Edition, Springer Verlag, N.Y., 1986
- [9] Kohn, W., "A Rendezvous Expert System for STS," ROBEX, Pittsburgh, PA, July 1987
- [10] Kohn, W., Skillman, T., "Hierarchical Control Systems for Autonomous Space Robots," Proc. of GNC, AIAA 88, Minneapolis, Minnesota, August 15-17, pp. 382-390