COMPOSITIONAL MODELING FOR
SPATIAL PROBLEMS

BY KYUNGSOOK HAN

A dissertation submitted to the
Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Graduate Program in Computer Science

Written under the direction of
Professor Andrew Gelsey
and approved by

__________________________

__________________________

__________________________

__________________________

New Brunswick, New Jersey
May, 1994
ABSTRACT OF THE DISSERTATION

Compositional Modeling for Spatial Problems

by Kyungsook Han

Dissertation Director: Professor Andrew Gelsey

Solving a problem about a complex physical system generally involves the creation and execution of a model needed to reason about the problem. Effective problem solving about a physical system requires the use of an adequate model, the creation of which in turn depends on the types of knowledge available for the physical system and their representation. Such a model is normally created by the person studying the system, but a hand-crafted model is often error-prone. Modifying a hand-crafted model to solve a similar problem about other physical systems is also difficult, and may take more time than building a new model for the systems.

My research has two main goals: (1) automating the construction and execution of models of physical systems for spatial problems, where objects are related to each other either geometrically or topologically to satisfy a set of constraints on the physical systems; and (2) making the modeling process general so that common domain theories can be shared and reused instead of being duplicated. This dissertation focuses on two subclasses of spatial problems – problems about
the physical phenomena of motion of mechanical systems and RNA secondary structure (folding).

I have made important progress in automating the model-building and simulation process for the physical phenomena by developing new methods which use knowledge from physics laws, biological principles, and other sources. The methods have a common characteristic framework at high level which analyzes a problem, searches for model fragments relevant to the problem, constructs a model with them, and applies the model to solve the problem. The model fragments for the model of motion represent knowledge in a purely declarative, algorithm-independent form; most knowledge is just the same fundamental equations that appear in any standard text on the subject. The model fragments for the model of folding are structural elements. The methods have been implemented in working programs and tested in the domains of mechanical devices, sailboats, and RNA molecules. Experimental results show that the methods are capable of automatically generating correct behavioral or structural models of several different types of physical systems which involve motion or folding.
Acknowledgements

My advisor Andrew Gelsey has played a significant role in my thesis research. I am indebted to him for taking me on as his student, guiding me at every point in the thesis research, many hours reading multiple drafts of this thesis and the papers we coauthored, and insightful and detailed comments on them. He also provided me with financial support during summer 1993 (under the National Science Foundation grant CCR-9209793).

I am grateful to Chitoor Srinivasan for kindly agreeing to join my thesis committee on short notice and providing valuable comments on my thesis draft. He led me to abandon earlier format of the thesis draft, which has greatly improved the clarity of the final thesis.

Don Smith also kindly agreed to join my thesis committee and provided valuable comments on my thesis draft as well as the important issues and directions for further research.

Alexander Kister has also played an important role in my work on RNA folding problems. His work on the structure formation process inspired me to seek a simpler method to model the process. Although he is far away from Rutgers, he has been always prompt and kind in responding to my emails.

I would also like to thank the Department of Computer Science for providing me with teaching assistantship during my graduate study at Rutgers.

My husband, Hong-Jin, taught me a lot of basic concepts of molecular biology when I first started working on the problems in that area. He encouraged and supported me during the difficult times of my graduate career and did many house
chores for me while I worked on my thesis. His constant love and prayers have made this research possible.

I wish to thank my parents for raising me with love and sending me many things from Korea during my graduate study in the United States. I am indebted to my sister Myungsook and her husband for their prayers and support over the years. They played role models as born-again Christians.

I am especially appreciative of Jesus Christ, who is my life, for taking my Ph.D. from vision to reality.
# Table of Contents

Abstract ................................................................. ii
Acknowledgements .................................................... iv
List of Tables .......................................................... ix
List of Figures .......................................................... x

1. Introduction .......................................................... 1
   1.1. Problems and Motivation ...................................... 1
   1.2. Modeling Motion of Mechanical Systems ................... 2
   1.3. Modeling RNA Secondary Folding Structure ................. 4
   1.4. Guide to the Reader ........................................... 6

2. Framework for Modeling and Simulation .......................... 8
   2.1. Models for Spatial Reasoning Tasks ......................... 8
   2.2. The Epistemology of Physical System Modeling ............ 9
   2.3. Ontology and Representation ................................ 10

3. Modeling Motion .................................................... 14
   3.1. Background .................................................... 14
       3.1.1. Reference Frames ...................................... 14
       3.1.2. Euler Angles .......................................... 16
       3.1.3. Equations of Motion and Forces ....................... 19
       3.1.4. Differential Equation and Numerical Simulation .... 22
3.2. Problem Formulation ........................................ 25
3.3. Algorithm .................................................. 27
3.4. Examples .................................................... 36
   3.4.1. Single Spring-Block System .............................. 36
   3.4.2. Multiple Spring-Block System ......................... 46
   3.4.3. Sailboat in Fluid ..................................... 54
   3.4.4. Disc Mounted on a Rotating Shaft .................... 66
3.5. Related Research .......................................... 68
   3.5.1. Model Generation ..................................... 68
   3.5.2. Simulator Generation ................................. 70
   3.5.3. Other AI Research on Physics ......................... 71
   3.5.4. Model Selection ....................................... 73
   3.5.5. Model Approximation ................................ 74
   3.5.6. Model-Based Application ............................. 75
3.6. Directions for Further Research .......................... 75
3.7. Summary .................................................. 85

4. Modeling Folding ........................................... 87
4.1. Background ............................................... 88
   4.1.1. RNA .................................................. 88
   4.1.2. Nomenclature of RNA structures ....................... 89
   4.1.3. Existing Theoretical Approaches to RNA Structure Prediction ................................ 91
4.2. Problem Formulation .................................... 94
4.3. Algorithm ............................................... 95
4.4. Examples ............................................... 105
   4.4.1. tRNA ................................................ 105
4.4.2. TAR of HIV-1 ........................................ 112
4.4.3. cis-acting packaging sequences of HIV-1 ............ 117
4.5. Experiments with Random Sequences ....................... 122
4.6. Extension to Broader Class of Physical Systems .......... 123
4.7. Related Research ........................................ 126
4.8. Directions for Further Research .......................... 128
4.9. Summary .................................................. 129

5. Conclusions .................................................. 131
5.1. Summary .................................................. 131
5.2. Review of Thesis Goals ................................... 132
5.3. Contributions ............................................. 135
5.3.1. Contributions of ORACLE .............................. 135
5.3.2. Contributions of FOLDER .............................. 137

Appendix A. Model Fragments of ORACLE ....................... 138

Appendix B. Entity Types of ORACLE ............................ 143

Appendix C. Other Experimental Results of FOLDER ............ 156

C.1. Prokaryotic 5S rRNA .................................. 156
C.2. RRE of HIV-1 ........................................... 156
C.3. 16S rRNA ................................................. 160

References ..................................................... 167
Vita ............................................................. 175
List of Tables

3.1. Size of a model and time for formulating and solving it for each of
the Oracle examples. The units of the CPU times are in
seconds. ......................................................... 80

4.1. Types of base pairings between two bases ....................... 89

4.2. The Loci, definitions, and accession numbers of tRNA sequences
used for the experiment ........................................... 107

4.3. Covariation analysis of the helices in Figures 4.13 and 4.14 ...... 120

4.4. Number of potential helices in the covariation analysis of random
sequences .......................................................... 124

4.5. Lengths of potential helices in the covariation analysis of similar
random sequences and homologous RNA sequences ............... 124
List of Figures

1.1. (a) Linear harmonic oscillator (b) A single spring-block system . . 2
1.2. *Stars & Stripes '87* ............................................................... 3
1.3. Secondary structure of RNA ..................................................... 5
3.1. Reference frames $F$ and $F'$ .................................................. 14
3.2. The rotations defining the Euler angles, after [Goldstein, 1965] . . 17
3.3. The problem statement data structure .................................... 25
3.4. The model data structure ....................................................... 25
3.5. The entity data structure ....................................................... 27
3.6. The model fragment data structure ....................................... 28
3.7. A single spring-block system ............................................... 36
3.8. Instance of the problem statement data structure for the single
     spring-block system .............................................................. 37
3.9. Maple procedure attached to the if_needed facet of the velocity slot 43
3.10. Maple procedure attached to the if_needed facet of the ang_velocity
      slot ................................................................. 40
3.11. Motion of the block b1. Spring not shown. ............................. 43
3.12. Plots of the 12 state variables of the block b1 of the single spring-
      block system as functions of time ..................................... 44
3.13. The kinetic, potential, and total energy of the single spring-block
      system as a function of time during the simulation ................. 45
3.14. 2 blocks connected by 1 spring. Both blocks are pulled in opposite
      directions and released ..................................................... 46
3.15. 3 blocks connected by 2 springs. The middle block is pulled directly
to the side and released. ................................. 46
3.16. Input to the two-block system .......................... 47
3.17. Motion of the two-block system with a spring of nonzero damping
coefficient. Spring not shown. ............................. 48
3.18. Motion of the three-block system with springs of zero damping
coefficients ................................................. 49
3.19. Motion of the three-block system with springs of nonzero damping
coefficients ................................................. 49
3.20. Plots of the 12 state variables of the middle block of the three-block
system with springs of zero damping coefficients as functions
of time .......................................................... 50
3.21. The kinetic, potential, and total energy of the three-block system
as a function of time during the simulation for the case with
zero damping coefficient ........................................ 51
3.22. Plots of the 12 state variables of the middle block of the three-
block system with springs of nonzero damping coefficients as
functions of time ................................................. 52
3.23. The kinetic, potential, and total energy of the three-block system
as a function of time during the simulation for the case with
nonzero damping coefficient ................................. 53
3.24. *Stars & Stripes*, winner of the 1987 America’s Cup competition 54
3.25. Directions of force components on a sailboat, adapted from [Letcher,
1976] .......................................................... 55
3.26. Input for the sailboat problem ............................ 58
3.27. Input for the algebraic model of the sailboat problem ............. 60
3.28. Input for the differential model of the sailboat problem ........... 61
3.29. The sailboat speed as a function of time .................. 62
3.30. A speed polar showing the boat speed $V_s$ as a function of the course angle $\alpha$ for the wind speed 16.9 ft/sec. The negative x-axis is the wind direction and the positive x-axis is the upwind direction. The course angle is measured from the upwind direction. The radius from the origin is the boat speed. . . 63
3.31. Algebraic model formulated by Oracle for the sailboat problem (edited for readability) .................. 64
3.32. Differential model formulated by Oracle for the sailboat problem (edited for readability) .................. 65
3.33. Disc mounted on a rotating shaft .................. 66
3.34. Input for the disc-rod problem .................. 67
3.35. Size of a model as a function of #unknowns in the equations of the model. For the count of lines of a model with 3 unknowns, the average value of the line counts of the 3 models with 3 unknowns (disc-rod, algebraic model of boat, differential model of boat) is used. .................. 81
3.36. Total CPU time in seconds for generating a model and solving it as a function of #unknowns in the equations of the model. For the time of a model with 3 unknowns, the average value of CPU time of the 3 models with 3 unknowns (disc-rod, algebraic model of boat, differential model of boat) is used. . 81
3.37. Comparison of the size growth of a model with the time growth for formulating and executing a model. The units of the model sizes are in the line counts of models and the units of the times are in seconds. .................. 83
4.1. Terminology of elementary structure motifs of secondary structure 90
4.2. The alignment data structure ........................................... 95
4.3. The secondary structure data structure ............................... 96
4.4. The diagonal data structure ........................................... 101
4.5. Alignment of 10 tRNA sequences ..................................... 106
4.6. Covariation matrix obtained as a result of comparative analysis on
     the 10 tRNA sequences .............................................. 108
4.7. tRNA structure inferred from the covariation matrix ............. 109
4.8. tRNA structure drawn by LoopViewer given the input of a struc-
     ture in a form of matching parentheses ............................ 109
4.9. Model of the structure formation process of tRNA .................. 110
4.10. (a) Alignment of 10 TAR RNAs of HIV-1 (b) Covariation matrix 114
4.11. Possible 3 structures of TAR ....................................... 115
4.12. Model of the folding process of TAR ................................ 116
4.13. A plausible structure for 10 HIV-1 cis-acting packaging se-
     quences ..................................................................... 118
4.14. Another plausible structure for 10 HIV-1 cis-acting packaging se-
     quences ..................................................................... 118
4.15. An optimal structure of $\text{HIV}_{LAI}$ packaging sequences (base 227 –
     base 335 of $\text{HIV}_{LAI}$ genomic sequence) predicted by Zuker
     and Stiegler's folding program [1981]. SD stands for splice
donor site. ................................................................. 121
4.16. An optimal structure of $\text{HIV}_{NL43}$ packaging sequences (base 227 –
     base 335 of $\text{HIV}_{NL43}$ genomic sequence) predicted by Zuker
     and Stiegler's folding program ....................................... 122
B.1. Taxonomy of entities of ORACLE ...................................... 143
C.1. Alignment of 34 prokaryotic 5S rRNAs .............................. 157
C.2. Covariation matrix of the 34 Prokaryotic 5S rRNAs, generated by
C.3. Common structure of the 34 Prokaryotic 5S rRNAs . . . . . . . . . . 160
C.4. Common structure of 10 RRE sequences of HIV-1 . . . . . . . . . . 161
C.5. Common structure of the 5' domain of 10 16S rRNAs . . . . . . . . 163
C.6. Common structure of the central domain of 10 16S rRNAs . . . . 164
C.7. Common structure of the 3' major domain of 10 16S rRNAs . . . 165
C.8. Common structure of the 3' minor domain of 10 16S rRNAs . . . 166
Chapter 1

Introduction

1.1 Problems and Motivation

This dissertation discusses the problem of automating the model building process using model fragments to solve spatial problems about physical systems. The spatial problem is a problem about shape, position or motion. Spatial problems are involved in many areas of science and engineering, as well as our routine work in everyday life – design, computer graphics, route finding, furniture arrangement, and so on. This dissertation is about two classes of spatial problems: motion and folding, in particular, the motion of mechanical systems whose behavior can be described by a set of ordinary differential equations and the secondary structure of RNA molecules formed by folding. *Folding* of a molecule is the process involved in the conversion of the linear sequence of the molecule to higher-dimensional structures [Stenesh, 1989]. Both motion and folding are important to study because

1. They commonly occur in the physical world, and computer-assisted modeling of them plays an essential role at virtually every stage, theoretical and experimental, in making progress toward a better understanding of them.

2. There is no reliable and automatic method for modeling them for spatially complex problems, and therefore they present challenges which require new research.
Solving problems about complex physical systems generally involves the creation and execution of models needed to reason about the problems. Models are normally constructed by the person studying the system. Despite the considerable time and effort spent, a hand-crafted model is often error-prone. Modifying a hand-crafted model to solve similar problems about other physical systems is also difficult, and may take more time than building a new model for the systems. The work of this dissertation is motivated by two goals. The first goal is to automate the process of model-building and simulation of physical systems for spatial problems, where objects are related to each other either geometrically or topologically to satisfy a set of constraints on the physical systems. The second goal is to make the modeling process widely applicable to many different physical systems involving either motion or folding so that common domain theories can be shared and reused instead of being duplicated.

1.2 Modeling Motion of Mechanical Systems

Consider, for example, a spring with one end attached to a fixed point and the other end attached to a block, as illustrated in Figure 1.1a. This harmonic oscillator is a common textbook example which is often used in qualitative physics research. It is well known that the oscillator has one degree of freedom, i.e., displacement of the block from its equilibrium position, and its motion is oscillatory on a straight line. However, if you consider a block and a spring in more general

![Figure 1.1: (a) Linear harmonic oscillator (b) A single spring-block system](image-url)
configuration, predicting the behavior is not as simple as before. For example, the block in Figure 1.1b is pulled and rotated before being released. Is the motion going to be still oscillatory? More interesting questions include: (1) What if a spring is attached to a corner of a block instead of the center of a face? (2) What if multiple blocks connected by springs are put in arbitrary position and orientation? (3) Can we use the modeling system for spring-block systems to predict the behavior of another type of physical system, such as a sailboat shown in Figure 1.2, which appears quite different from the spring-block systems? All these problems will be addressed later in Chapter 3.

The motion of multiple moving objects in arbitrary configurations in three dimension is much more difficult to understand and deal with than the motion in one or two dimension, but is important because of its relevance to many practical applications, ranging from computer graphics to design, to mechanics problems. Modeling such a motion requires formulating equations in vector space and handling moving reference frames. There are several works on modeling motion, but spatially complex motions did not get much attention. For example, some qualitative physics approaches are used to model motion, but they focus on developing representations for physical systems and reasoning about the systems within the
representations, and are limited to simple motions. Some research in spatial reasoning or commercial mechanics simulators have powerful algorithms, but they incorporate knowledge of physical phenomena directly into algorithms and difficult to reuse or extend them to solve similar problems about other physical systems. Some works assume the existence of a model, and concentrate on transforming the model into an executable simulation code instead. As we will show later in Chapter 3, we handle explicitly vector quantities and reference frames, and construct a model using a set of model fragments containing equations. The model fragments represent the fundamental physics knowledge in a declarative and algorithm-independent way, and can be reused in building behavioral models of different types of physical systems.

1.3 Modeling RNA Secondary Folding Structure

Ribonucleic acid (RNA) molecules are usually single-stranded molecules, consisting of four different types of bases. The single-stranded RNA molecules are not found in nature as simple chains. In their biologically active form, they are folded (i.e., bended) back upon themselves, forming complex structures (see Figure 1.3 for an example). RNA folding structure is important because it determines the biological function of RNA.

For an RNA of $N$ bases, the total number of possible structures is exponential in $N$. An exhaustive search for every possible structure is inefficient and intractable for this problem. The folding structure can be determined directly by X-ray crystallography, but most RNAs cannot be crystallized with current technique. Many experiments have been carried out with chemically synthesized RNAs to obtain the stability of each structure element measured by the loss in free energy in kcal/mol. A data set of free energy values describing the structural stability is called an energy model. Several energy models are available but none
Figure 1.3: Secondary structure of RNA
of them can be considered perfect. Predicted structures are often different if a
different energy model is used or a slight change is made to the energy values.

Most work on RNA structure prediction exists outside of AI. The energy min-
imization method uses an energy model to determine a folding structure with
minimum free energy. Limitations of this method are partly due to the uncer-
tainty of the underlying energy model itself. As we will show later, this method
is also sensitive both to point mutation and to range of a sequence. The phy-
logenetic comparative method examines, usually manually, related sequences to
identify potential helices, which maintain complementarity in the sequences. An
existing algorithm for simultaneously aligning and folding multiple sequences re-
quires exponential time and space.

Since the energy model is believed to be inaccurate as well as incomplete, we
do not make commitment to the numeric values of the energy model. Instead,
we use approximation of the model and combine it with knowledge from other
sources, such as homologous RNAs. Homologous sequences are similar sequences
with same origin. We use a heuristic algorithm with folding constraint propagator,
which can predict a common folding structure of multiple sequences in polynomial
time and space.

1.4 Guide to the Reader

To obtain an overview of this dissertation, the reader should skim Chapters 1,
2, and 5. To understand more details about the method and the results of the
method, Chapters 3 and 4 should be read closely. These chapters are organized
as follows:

Chapter 2 presents a high-level framework for constructing models and simu-
lationg them for two types of spatial problems, and describes ontology and
representation of the framework.
Chapter 3 describes the background knowledge required for modeling moving objects, and the details of the method to solve problems involving mechanical forces and motion. This chapter also describes experimental results of testing the method on several types of physical systems involving motion. It discusses other related research and identifies limitations of current work and directions for further research.

Chapter 4 describes the background knowledge required for modeling RNA folding structure, and the details of the method to solve problems involving folding. This chapter also describes experimental results of testing the method on various types of RNA sequences and random sequences, and other related work. It discusses limitations of current work and directions for further research.

Chapter 5 summarizes the main ideas and results of this dissertation, and discusses the contributions of the work of this dissertation.

Appendix A contains the model fragments currently available in Oracle, represented in the actual Maple\(^1\) syntax.

Appendix B describes the entity types currently available in Oracle, expressed in the actual Maple syntax.

Appendix C shows some other experimental results of testing the modeling system of folding on prokaryotic 5S rRNA, RRE of HIV-1, and 16S rRNA.

---

\(^1\)Oracle is implemented in the mathematical manipulation language Maple [Char et al., 1991].
Chapter 2
Framework for Modeling and Simulation

2.1 Models for Spatial Reasoning Tasks

What is spatial reasoning? There seems to be no widely agreed definition of it although spatial reasoning problems are considered in a variety of areas, including computer vision, computer graphics, robotics, and reasoning about physical systems. Lee and Hsu [1992], for instance, suggest the inference of a consistent set of spatial relationships among the objects as the meaning of spatial reasoning in the image information system. Following Charniak and McDermott [1987], we define it as reasoning about shape, position and motion.

Spatial reasoning is a central aspect of many areas which involve interactions with the physical world. But different areas have different tasks of spatial reasoning. In computer vision, for example, recognition of familiar objects can be the main spatial reasoning task. The spatial reasoning task we have in mind is to construct behavioral or structural models of physical systems which can be used in reasoning about the systems. Specifically, we focus on modeling moving mechanical systems and on modeling RNA folding structures.

There are many tasks which require producing behavioral models of physical systems. For example, behavioral models are necessary in diagnosis; diagnosis of an engineering system isolates faulty components of the system from the behavior difference between the behavior predicted by the model and the observed behavior of an actual system. Behavioral models can also be used as part of the design
process in testing a candidate design solution by generating a behavioral model and comparing it with the desired behavior.

Structural models of RNAs are essential in understanding RNA because the folding structure of RNA determines its biological function. This is why many biologists want to know the structure of an RNA molecule they are working on. In the long run, progress in predicting RNA structures will accelerate finding ways to prevent or cure disease caused by RNA viruses, which include AIDS, cold, influenza, and hepatitis.

2.2 The Epistemology of Physical System Modeling

When a model of a complex system, both human-designed and natural, is reported in the literature, there is a considerable emphasis on the end result, the model. The construction of a model and its use is a difficult problem, and it often requires the use of many types of information. On the other hand, many works on simulation assume the existence of models of physical systems, and focus on developing representations for the physical systems and reasoning about the models in the representations. However, not every physical system has its models ready to use for problem-solving tasks and constructing adequate models requires considerable human effort. Besides, choosing a simulation method is often dependent on the types of models available. If an exact model cannot be found, we may have to rely on approximate models, thus restricting possible simulation methods to use. Even if an exact quantitative model exists, it may be preferable to have an approximate model when the model or its solution is too complex to be comprehensible. If the simulation result of an approximate model turns out to provide insufficient information for a problem-solving task, we might want to consider a more detailed model and its simulation method.

In this dissertation we examine the process of model building and simulation,
as well as the types of knowledge required to perform the process of modeling and simulation or to evaluate the process and its results. We also discuss the epistemological adequacy of a modeling system for broader class of physical systems, and address related issues such as: Can a modeling system of a physical system be used to model different physical systems? Do we need a separate modeling system for every physical system? Or the same modeling system just with more knowledge will suffice to model different physical systems?

2.3 Ontology and Representation

The principal elements of our ontology are entities, physical systems, phenomena, model fragments, and models. An entity is a physical object which either constitutes a physical system by itself (i.e., primitive object) or is a part of a physical system (i.e., composite object). The block entity, for example, has properties such as position and velocity. An entity is represented in a frame with slots for the properties. Facets allowed in a slot are value, form, range, if_needed, and if_added. The value facet is initially set to null but will be assigned a vector, scalar, string, set, or any other expression as it becomes known. The form facet distinguishes the slot type (e.g., scalar or vector) and is consulted when the system creates a new variable name during the problem solving process. For example, if the system is asked to compute the position of a block b1, a set of new variables \{b1x(t), b1y(t), b1z(t)\} will be created for the position vector and used in equations. The range facet specifies a valid range of the property value if it is known. The if_needed facet or if_added facet holds the procedure call, invoked when a slot value is needed or added. The if_needed procedure of the velocity slot in the example below says that velocity is derivable from position.

```
block=[AKO=rigidbody_with_constmass,
      position(t)=[value=null, form=[x(t),y(t),z(t)]],
```
\[ \text{velocity}(t)=[\text{value=\text{null}}, \text{form}=[u(t), v(t), w(t)], \]
\[ \text{if\_needed}=[\text{derive\_velocity, position}(t)], \]
\[ \ldots \text{ (other slots not shown)} \ldots ] \]

A ribonucleotide, a building block of RNA, can be represented as follows:

\[ \text{ribonucleotide}=[\text{AKO=nucleotide,} \]
\[ \text{base}=[\text{value=\text{null}}, \]
\[ \text{position\_in\_sequence}=[\text{value=\text{null}}, \]
\[ \text{paired\_base}(t)=[\text{value=\text{null}}] \]

A physical system is a collection of one or more entities. The structure of a physical system is determined by its constituent entities and their connections. A spring-block system, for example, is a composite object with spring(s) and block(s) as its components. An RNA molecule is a composite object made up of ribonucleotides.

A phenomenon is a process which changes one or more properties of an entity in a physical system. Force from a spring, for example, is a phenomenon which can change the position and/or orientation of an object which is attached to the spring. Base pairing, for example, is a phenomenon which can change the connection (i.e., paired\_base) of a ribonucleotide.

A model fragment is a partial description of a physical phenomenon by a set of entities, variables, assumptions, and/or equations. There may be more than one model fragments for a single phenomenon, each with different assumption or approximation. A model fragment is applicable when the corresponding phenomenon occurs. The spring force, for example, exerted on an object attached to end2 of a linear spring with linear damping is represented as follows:\footnote{The representation of the model fragment is edited for clarity. See Appendix A for the actual syntax of the model fragment.}:

---
Springforce2=[phenomenon='spring force at end2',
    entities=[s=linear_damped_spring],
    variables=[k=s[force_const], b=s[damping_coeff],
               e1(t)=s[end1(t)], e2(t)=s[end2(t)],
               l=s[rest_length], f(t)=s[force2(t)],
    equations=[f(t)=-k*(||e2(t)-e1(t)||-1)*(e2(t)-e1(t))/
               ||e2(t)-e1(t)||-b*diff(||e2(t)-e1(t)||-1)*
               (e2(t)-e1(t))/||e2(t)-e1(t)||, t]]

Note that the variables e1(t), e2(t), and f(t) are vectors and that the rest are scalars. The model fragment says that s is a linear damped spring, k is a force constant of the spring, b is a damping coefficient, e1(t) and e2(t) are the position vectors of end1 and end2, l is the rest length, and f(t) is the spring force at end2. ||e2(t)−e1(t)|| is the vector norm representing the length of the spring at time t, ||e2(t)−e1(t)||−1 is the signed length change from the rest length, and (e2(t)−e1(t))/||e2(t)−e1(t)|| is a unit vector with direction from end1 to end2. This representation is purely declarative, algorithm-independent. It is just the same fundamental equations with its implied semantics of vectors, which appear in any standard mechanics textbook.

A helix h formed by bases GGGCAAG in 1–7 and bases CUUGCCC in 79–85 of a tRNA r can be represented as follows. It says that h has S1 score of 2.5 and S2 score of 6.0, which will be explained later in section 4.3 (Slightly different syntax is used in the implementation of modeling folding structures).

Basepairing=[phenomenon='base pairing by hydrogen bonds',
    entities=[r=tRNA, h=helix],
    variables=[GGGCAAG=r[1-7], CUUGCCC=r[79-85],
               s1=h[S1_score], s2=h[S2_score]],
    helix=[h=GGGCAAG/CUUGCCC, s1=2.5, s2=6.0]]
A model is a composition of model fragments applicable to a physical system in a particular situation. Simulation is the execution of a model.

Both motion and folding can change the structure of a physical system in which they occur. However, the ways they change the structure are different. Motion can change the structure of a physical system by changing the positions, orientations, or contact relations of its components. But it usually leaves the rigid connectivity relations of the components intact. Folding makes a more drastic change to the structure by fundamental changes of the connections of the components as well as their positions, orientations, and contacts. Because of the dynamic connectivity between the components of a folding system, modeling the folding phenomenon focuses more on structure than modeling the moving phenomenon does.
Chapter 3
Modeling Motion

In this chapter we describe basic theories required to model motion, and present the modeling system of motion called ORACLE, implemented in the mathematical manipulation language Maple [Char et al., 1991].

3.1 Background

3.1.1 Reference Frames

Since we produce models of complex motions, coordinate systems are necessary in describing mechanical phenomena. A coordinate system is often called a frame of reference or reference frame. Pictured in Figure 3.1 are two rectangular cartesian coordinate systems. Suppose that $F$ is a stationary reference frame and that $F'$ is a reference frame fixed to an entity which moves relative to $F$. Let $r$ be the

![Diagram of reference frames](image)

Figure 3.1: Reference frames $F$ and $F'$
position vector of a point \( P \) in the entity, relative to the origin \( O \) of reference frame \( F \), and \( r' \) the position vector of \( P \) relative to the origin \( O' \) of \( F' \). \( R \) is the vector from \( O \) to \( O' \). The three vectors are related by

\[
\mathbf{r} = \mathbf{r'} + \mathbf{R}
\]

(3.1)

Each of the three vectors \( \mathbf{r} \), \( \mathbf{r'} \), and \( \mathbf{R} \) has a set of three components in each of the reference frames \( F \) and \( F' \). The components of a given vector in the two reference frames are related to one another by an orthogonal transformation (i.e., a transformation which does not involve a change of scale). Since the reference frame \( F' \) is fixed to a moving entity, \( F' \) moves as the entity moves. Thus, \( \mathbf{r'} \) is a fixed quantity. However, as the entity moves, \( \mathbf{R} \) changes and so does \( \mathbf{r} \).

The laws of motion are in general different in form for different reference frames. If an arbitrary reference frame is chosen, space would be inhomogeneous and anisotropic and the laws governing simple phenomena may have to be expressed in very complex forms [Landau and Lifshitz, 1976]. This means that, even if an object does not interact with any other objects, its various positions in space and its different orientations would not be mechanically equivalent. The same is in general true of time. For example, a free body which subject to no external action could not remain at rest; if its velocity were zero at some instant, it would begin to move in some direction at the next instant. However, it is found that we can choose a reference frame in which space is homogeneous and isotropic and time is homogeneous. This is called an \textit{inertial reference frame}. An inertial reference frame is a reference frame which is not rotating or accelerating [Barford, 1973; Bradbury, 1968]. If there exist several inertial reference frames, they move with constant velocities or are at rest, relative to each other. The axes need not be parallel, but will remain at constant angles with respect to each other. In such a reference frame a free body which is at rest at some instant remains always at rest.
A body reference frame or local reference frame is a reference frame fixed to
an entity (We will use the terms body reference frame and local reference frame
interchangeably in this thesis). To an observer in a body reference frame, the body
reference frame will appear stationary, although it may be regarded as moving
with or without acceleration when observed outside the body reference frame.
Notice that the inertial reference frame is not a word opposite in meaning to the
body reference frame. In this thesis, we will use the term “space reference frame”
as the antonym of the body reference frame, and “noninertial reference frame” as
the antonym of the inertial reference frame. Next sections will illustrate how to
transform from one reference frame to another.

3.1.2 Euler Angles

In this section we derive a representation for a transformation in terms of three
parameters which can be conveniently used to specify the orientation of a body.
One can carry out the transformation from a given cartesian coordinate system to
another by means of three successive rotations performed in a specific sequence.
The Euler angles are defined as the three successive angles of rotation.

The sequence will be started by rotating the initial system of axes, \(xyz\), by an
angle \(\phi\) counterclockwise about the \(z\) axis, and the resultant coordinate system
will be labeled the \(x_1y_1z_1\) axes. In the second state the intermediate axes, \(x_1y_1z_1\)
are rotated about the \(x_1\) axis counterclockwise by an angle \(\theta\) to produce another
intermediate set, the \(x_2y_2z_2\) axes. Finally the \(x_2y_2z_2\) axes are rotated counterclockwise by an angle \(\psi\) about the \(z_2\) axis to produce the desired \(x'y'z'\) system of
axes. Figure 3.2 shows the three stages of the sequence. The Euler angles \(\phi, \theta,\nand \(\psi\) thus completely specify the orientation of the \(x'y'z'\) system relative to the
\(xyz\) and can therefore act as the three needed generalized coordinates.

The elements of the complete transformation \(S\) can be obtained by writing
Figure 3.2: The rotations defining the Euler angles, after [Goldstein, 1965]
the matrix as the triple product of the separate rotations, each of which has a relatively simple matrix form. The initial rotation about $z$ can be described by a transformation matrix $S_1$.

\[
\begin{pmatrix}
  x_1 \\
y_1 \\
z_1
\end{pmatrix} = \begin{pmatrix}
  \cos \phi & \sin \phi & 0 \\
  -\sin \phi & \cos \phi & 0 \\
  0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
  x \\
y \\
z
\end{pmatrix} = S_1 \begin{pmatrix}
  x \\
y \\
z
\end{pmatrix} \tag{3.2}
\]

Similarly the transformation corresponding to a rotation about $x_1$ is described by $S_2$.

\[
\begin{pmatrix}
  x_2 \\
y_2 \\
z_2
\end{pmatrix} = \begin{pmatrix}
  1 & 0 & 0 \\
  0 & \cos \theta & \sin \theta \\
  0 & -\sin \theta & \cos \theta
\end{pmatrix} \begin{pmatrix}
  x_1 \\
y_1 \\
z_1
\end{pmatrix} = S_2 \begin{pmatrix}
  x_1 \\
y_1 \\
z_1
\end{pmatrix} \tag{3.3}
\]

Finally, rotation by $\psi$ about the $z_2$ axis is described by a matrix $S_3$

\[
\begin{pmatrix}
  x' \\
y' \\
z'
\end{pmatrix} = \begin{pmatrix}
  \cos \psi & \sin \psi & 0 \\
  -\sin \psi & \cos \psi & 0 \\
  0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
  x_2 \\
y_2 \\
z_2
\end{pmatrix} = S_3 \begin{pmatrix}
  x_2 \\
y_2 \\
z_2
\end{pmatrix} \tag{3.4}
\]

The overall transformation from the space reference frame to the body reference frame is found by combining the three matrices:

\[
\begin{pmatrix}
  x' \\
y' \\
z'
\end{pmatrix} = S_3 S_2 S_1 \begin{pmatrix}
  x \\
y \\
z
\end{pmatrix} = S \begin{pmatrix}
  x \\
y \\
z
\end{pmatrix} \tag{3.5}
\]

By performing the matrix multiplication $S = S_3 S_2 S_1$ we find

\[
S =
\begin{pmatrix}
  \cos \psi \cos \phi - \cos \theta \sin \phi \sin \psi & \cos \psi \sin \phi + \cos \theta \cos \phi \sin \psi & \sin \psi \sin \theta \\
-\sin \psi \cos \phi - \cos \theta \sin \phi \cos \psi & \sin \psi \sin \phi + \cos \theta \cos \phi \cos \psi & \cos \psi \sin \theta \\
\sin \theta \sin \phi & \sin \theta \cos \phi & \cos \theta
\end{pmatrix}
\tag{3.6}
\]
The transformation $S$ is a function of the three independent parameters $\phi$, $\theta$ and $\psi$ of the Euler angles. The inverse transformation from the local coordinates to space axes

$$
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix} = S^{-1} \begin{pmatrix}
x' \\
y' \\
z'
\end{pmatrix}
$$

is then given by the inverse matrix $S^{-1}$, which is identical with the transposed matrix $S^t$ because $S$ is an orthogonal matrix:

$$
S^{-1} = S^t =
\begin{pmatrix}
\cos \psi \cos \phi - \cos \theta \sin \phi \sin \psi & - \sin \psi \cos \phi - \cos \theta \sin \phi \cos \psi & \sin \theta \sin \phi \\
\cos \psi \sin \phi + \cos \theta \cos \phi \sin \psi & - \sin \psi \sin \phi + \cos \theta \cos \phi \cos \psi & - \sin \theta \cos \phi \\
\sin \theta \sin \psi & \sin \theta \cos \psi & \cos \theta
\end{pmatrix}
$$

(3.8)

### 3.1.3 Equations of Motion and Forces

In the general motion of an entity involving both translation and rotation, there are a total six degrees of freedom. A total of six coordinates are needed and, for example, there could be three components for the position $\mathbf{r}$ of the entity and the three components for the orientation $\mathbf{q}$. We use the three rectangular coordinates of the center of mass for the position, and the three Euler angles for the orientation.

$$
\mathbf{r} = \begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
$$

(3.9)

$$
\mathbf{q} = \begin{pmatrix}
\phi \\
\theta \\
\psi
\end{pmatrix}
$$

(3.10)
The velocity vector can then be defined formally by the equation:

\[ \mathbf{v} = \frac{d\mathbf{r}}{dt} \tag{3.11} \]

That is, in component form,

\[ u = \frac{dx}{dt}, \quad v = \frac{dy}{dt}, \quad w = \frac{dz}{dt} \tag{3.12} \]

The angular velocity vector \( \mathbf{\Omega} \) can also be expressed in terms of the Euler angles and their time derivatives. Referring to Figure 3.2a, the rotation \( \phi \) produces an angular velocity \( \dot{\phi} \) along the \( z \) axis. In the \( x_1y_1z_1 \) coordinate system, this is represented by

\[ (0, 0, \dot{\phi}) \tag{3.13} \]

The rotation \( \theta \) in Figure 3.2b produces a component of angular velocity \( \dot{\theta} \) in the \( x_2 \) direction. Resolution of \( \dot{\phi} \) into its \( y_2 \) and \( z_2 \) components yields the combined effects of the \( \theta \) and \( \phi \) rotations expressed in the \( x_2y_2z_2 \) coordinate system:

\[ (\dot{\theta}, \dot{\phi}\sin \theta, \dot{\phi}\cos \theta) \tag{3.14} \]

The rotation \( \psi \) in Figure 3.2c produces a component \( \dot{\psi} \) in the \( z' \) direction. The combined effects of all three rotations in the body reference frame is then

\[ \Omega'_1 = \dot{\phi}\sin \theta \sin \psi + \dot{\theta} \cos \psi \]
\[ \Omega'_2 = \dot{\phi}\sin \theta \cos \psi - \dot{\theta} \sin \psi \tag{3.15} \]
\[ \Omega'_3 = \dot{\phi}\cos \theta + \dot{\psi} \]

The components of \( \mathbf{\Omega} \) in the space reference frame can be obtained by means of the vector transformation

\[ \mathbf{\Omega} = S^t \mathbf{\Omega'} \tag{3.16} \]

The result is as follows.

\[ \Omega_1 = \dot{\psi}\sin \theta \sin \phi + \dot{\theta} \cos \phi \]
\[ \Omega_2 = -\dot{\psi}\sin \theta \cos \phi + \dot{\theta} \sin \phi \tag{3.17} \]
\[ \Omega_3 = \dot{\psi}\cos \theta + \dot{\phi} \]
The linear momentum $\mathbf{p}$ of an entity is defined in terms of the velocity and mass.

$$p = mv$$  \hspace{1cm} (3.18)

The angular momentum $\mathbf{M}$ is defined by the cross product of the position vector and the linear momentum.

$$\mathbf{M} = \mathbf{r} \times \mathbf{p}$$  \hspace{1cm} (3.19)

The motion of an entity at any instant can be described by a set of twelve differential equations in the twelve components of four vectors: position, orientation, velocity, and angular velocity. Six of them are found in Equations 3.12 and 3.17, which say that the velocity and the angular velocity functions can be derived by differentiating the position and the orientation functions, respectively. Three more differential equations are obtained by the time derivative of the momentum vector:

$$\frac{d\mathbf{p}}{dt} = \mathbf{F}$$  \hspace{1cm} (3.20)

where $\mathbf{F}$ is the total force ($\mathbf{F} = \sum \mathbf{f}$) acting on the entity and $\mathbf{p}$ is the momentum of it. Equation 3.20 represents a fundamental postulate in mechanics which is commonly known as Newton's second law of motion. If the total force $\mathbf{F}$ is zero, then $d\mathbf{p}/dt = 0$ and the momentum $\mathbf{p}$ is constant.

The last three equations of motion come from Euler's equations of motion:

$$I_1 \dot{\Omega}_1 + (I_3 - I_2) \Omega_2 \Omega_3 = K_1$$

$$I_2 \dot{\Omega}_2 + (I_1 - I_3) \Omega_3 \Omega_1 = K_2$$  \hspace{1cm} (3.21)

$$I_3 \dot{\Omega}_3 + (I_2 - I_1) \Omega_1 \Omega_2 = K_3$$

where $I_1$, $I_2$, and $I_3$ are the principal moments of inertia of inertial tensor, $\Omega_1$, $\Omega_2$, and $\Omega_3$ are the three components of the angular velocity, and $K_1$, $K_2$, and $K_3$ are the three components of the total torque defined as:

$$\mathbf{K} = \sum \mathbf{r} \times \mathbf{f}$$  \hspace{1cm} (3.22)
Therefore, the motion of an entity can be described by twelve differential equations in Equations 3.12, 3.17, 3.20, and 3.21 with total six degrees of freedom. Similarly, the motion of a physical system with \( n \) subparts can be characterized by 12\( n \) differential equations with maximum 6\( n \) degrees of freedom.

### 3.1.4 Differential Equation and Numerical Simulation

For motion of physical systems, we focus on behavioral models which are best represented as sets of equations. Equations specify constraints over the values of variables. For the types of equations we consider ordinary differential equations (ODE) and algebraic equations. If Maple can solve the equations analytically, their solutions are obtained in closed forms. However, differential equations are often nonlinear and do not have a solution in closed form, and must be solved by numeric integration. For the purpose of numeric integration, differential equations are required to be in the **canonical form**:

\[
\frac{ds_i(t)}{dt} = f_i(t, s_1, \ldots, s_i, \ldots, s_m)
\]  
(3.23)

for \( a \leq t \leq b \), with the initial condition of a state variable \( s_i \)

\[
s_i(a) = \alpha_i
\]  
(3.24)

A higher-order differential equation can always be transformed into a system of first-order differential equations in the canonical form by trivial introducing new variables. For example, the second-order equation \( \frac{d^2a(t)}{dt^2} = a(t) \) can be reduced to a system of two first-order equations by introducing a new variable \( v \):

\[
\begin{align*}
\frac{dx(t)}{dt} &= v(t) \\
\frac{dv(t)}{dt} &= a(t)
\end{align*}
\]  
(3.25)

If a behavioral model of a physical system is expressed by a set of differential equations in the form of Equation 3.23, the behavior is simulated by numerically
Algorithm 3.1 Runge-Kutta method of order four for systems of differential equations

\[ h = (b-a)/N; \quad t = a \]

for \( i = 1, 2, \ldots, m \)
\[ w_i = \alpha_i \]

OUTPUT \((t, w_1, w_2, \ldots, w_m)\)

for \( j = 1, 2, \ldots, N \) do
  for \( i = 1, 2, \ldots, m \)
    \[ k_{1,i} = hf_i(t, w_1, w_2, \ldots, w_m) \]
  for \( i = 1, 2, \ldots, m \)
    \[ k_{2,i} = hf_i(t + h/2, w_1 + 1/2k_{1,1}, w_2 + 1/2k_{1,2}, \ldots, w_m + 1/2k_{1,m}) \]
  for \( i = 1, 2, \ldots, m \)
    \[ k_{3,i} = hf_i(t + h/2, w_1 + 1/2k_{2,1}, w_2 + 1/2k_{2,2}, \ldots, w_m + 1/2k_{2,m}) \]
  for \( i = 1, 2, \ldots, m \)
    \[ k_{4,i} = hf_i(t + h, w_1 + k_{3,1}, w_2 + k_{3,2}, \ldots, w_m + k_{3,m}) \]
  for \( i = 1, 2, \ldots, m \)
    \[ w_i = w_i + (k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i})/6 \]
  \[ t = a +jh \]

OUTPUT \((t, w_1, w_2, \ldots, w_m)\)

solving these differential equations for \( s_i(t) \) at successive time steps, starting with a given initial state \( s_i(a) \). The accuracy of the solution typically increases as the stepsize decreases. There are many methods for numerically solving ordinary differential equations. We compute numerical solutions by implementing Runge-Kutta order four method in Maple. A system of \( m \) first-order differential equations expressed in the form

\[ \frac{ds_i(t)}{dt} = f_i(t, s_1, s_2, \ldots, s_m), \quad i = 1, 2, \ldots, m \]

\[ a \leq t \leq b, \quad s_i(a) = \alpha_i, \quad i = 1, 2, \ldots, m \]

can be solved by the algorithm of the fourth-order Runge-Kutta method [Burden and Faires, 1985] shown in Algorithm 3.1.

The input to the algorithm consists of end points \( a \) and \( b \), number of time
steps, a set of differential equations, a set of variables, and their values at time $a$. The output of the algorithm is a set of values of the variables at intermediate time steps in the interval $[a, b]$. Since Maple performs computations by interpreting the user input, it is slower than a compiled code. Thus, we allow an alternative way to compute numerical solutions of differential equations using a publicly-available subroutine called rk4 [Press et al., 1988] which implements the same Runge-Kutta method in C. After Oracle constructs a model in Maple, it transforms the equations of the model into a C program, compiles the program, and executes the resulting code to perform simulation (The detailed method for doing this appears in Section 3.4.1). When the same step size and precision are used, the result of the simulation by a compiled code is as accurate as that by a Maple procedure, but is much faster. Oracle's implementation of the fourth-order Runge-Kutta method in Maple is general and can be used in any problem which needs numerical solutions of ODE. However, the way of transforming Maple models into C programs is not flexible and is restricted to the problems of spring-block systems in the current implementation of Oracle because simulating a model in an efficient method is not the focus of this work.

Some physical systems reach a steady state of behavior after a certain point in time. The behavior of the physical systems in such a steady state can be described by algebraic equations instead of ordinary differential equations because the time-derivatives of all the variables are zero, and the equations can be solved by algebraic methods. Section 3.4.3 has a detailed example of solving a problem both by the differential equation model and by the algebraic equation model.
**Oracle Problem Statement:**

1. list of entities whose elements have the following structure
   (a) entity name
   (b) entity class
   (c) list of pairs of property name and its (initial) value
2. list of constraints, each of which is of type either equality, inequality, maximize, minimize, or function specifying a problem solving method
3. list of variables, each of the form
   entity.name[property.name]

Figure 3.3: The problem statement data structure

**Oracle Model:**

1. set of variables
2. set of assumptions
3. set of model fragments instantiated to construct the model
4. list of equations

Figure 3.4: The model data structure

### 3.2 Problem Formulation

The kinds of problems solved by Oracle are defined as follows.¹

**Given:** a description of a physical system in the form of Figure 3.3

**Find:** a behavioral model described in the form of Figure 3.4 to predict the motion of the system and the variable values obtained by executing the model.

The problem statement of Oracle describes the entities of a physical system, any constraints to be satisfied, and the properties of the entities (i.e., variables)

¹Although many problems can be cast in this form, it does not mean Oracle can actually solve all such problems. The types of physical systems whose motions can be modeled by Oracle are restricted to the systems whose behavior can be described by a set of ordinary differential equations.
whose values are to be sought during modeling and simulation. The model produced by ORACLE contains variables, any modeling assumptions used, names of model fragments instantiated for the model, and equations. The equations of the model are solved either analytically or by numeric simulation to compute the values of the variables requested in the problem statement.

The most basic types of motion are translation and rotation. Most of other variations such as oscillate, fall, roll, and slide are the combined occurrences of translations and/or rotations in certain ways.

As discussed earlier, the motion of an entity at any instant can be described by a set of ordinary differential equations in the twelve components of four vectors: position, orientation, velocity, and angular velocity. The differential equations are often nonlinear and do not have a solution in closed form, and must be solved by numeric integration. For a moving entity, ORACLE constructs a model with the four vectors (position, orientation, velocity, and angular velocity) as state variables, which take numeric values during simulation.

The state variables of each subpart of an entity are initially defined in the local reference frame, which is assumed to be fixed to the entity. Then each subpart defined in its local reference frame is translated and rotated by having its reference frame redefined in a common inertial reference frame. The system chooses the common inertial reference frame from local reference frames which are not accelerated. If there is no such reference frame (i.e., all the local reference frames are noninertial), it introduces a new inertial reference frame. If there are several inertial reference frames, the choice is arbitrary. This way of changing reference frames are often more useful than defining all subparts in the inertial reference frame and then transforming them to the desired place because (1) it is not trivial to initially define all subparts relative to each other, and (2) additional information may be specified for subparts in their local reference frames. Once
Oracle Entity:

1. entity class
2. AKO slot which specifies the super class of the entity
3. list of property slots, each of which has the following structure
   (a) property name
   (b) value facet
   (c) form facet
   (d) range facet specifying possible range of the value of the property if it is known
   (e) if_needed facet
   (f) if_added facet
4. mf slot which contains the names of potentially relevant model fragments

Figure 3.5: The entity data structure

A problem with subparts described in their local reference frames is given to Oracle, the system will redefine the subparts in the inertial reference according to the locations and orientations of the local reference frames.

3.3 Algorithm

In this section we present the data structures and algorithms of Oracle. The knowledge base of Oracle contains entity classes and model fragments, represented in frame structures. The data structures of the entity and the model fragment are shown in Figure 3.5 and 3.6, respectively. Appendix A and B contain the model fragments and entity classes currently available in Oracle, represented in the actual Maple syntax. As illustrated in Figure B.1, the entity classes are organized in a tree. Properties of a class are inherited by its subclasses and instances.

The input to Oracle is a description of a physical system in terms of entities, constraints, and variables. See Figure 3.3 for the data structure of the input. As output, it produces a model of the motion of the system and the values of the
ORACLE Model Fragment:

1. phenomenon being characterized by the model fragment
2. list of entities, each of the form
   \( \text{entity\_name} = \text{entity\_class} \)
3. list of variables, each of the form
   \( \text{variable\_name} = \text{entity\_name}[\text{property\_name}] \)
4. set of assumptions
5. list of equations in component form (i.e., not in vector form)

Figure 3.6: The model fragment data structure

variable requested in the input statement. Figure 3.4 shows the data structure of the model. The model contains variable names, any modeling assumptions used, names of model fragments instantiated for the model, and equations. The variable values are obtained by solving the equations of the model either analytically or by numeric simulation.

As outlined in Algorithm 3.2, the top-level algorithm of ORACLE consists of three phases: (1) problem analysis; (2) model creation; and (3) model execution. In the first phase, ORACLE represents each entity of a problem statement in a frame by copying a class frame and filling in slots for property values specified in the problem statement. It also transforms vector quantities expressed in the local reference frames into those in the inertial reference frame, formulates initial conditions, and executes if-added procedures in the slots. A model fragment specifying forces on a component of a composite object is instantiated by if-added procedures in the this phase. After constraints are analyzed, variables are examined to determine if their values are already known in their slot values or derivable from other variables. In the second phase, additional model fragments which have not been instantiated are retrieved and a model is constructed from them. In the final phase, the constructed model is solved for the problem. If ORACLE runs out of potentially relevant model fragments before it finds a valid solution, it prints the
situation, asks more information, and quits. Since the number of model fragments of Oracle is finite, the algorithm is guaranteed to terminate. The remainder of this section describes each phase in more detail.

Given a problem statement in the form of Figure 3.3, Oracle analyzes the entities of the problem statement using Algorithm 3.3. It creates frame structures like Figure 3.5 for the entities and fills in slots for property values specified in the problem statement. When it fills in a slot value, it also checks whether there is an ifadded procedure in the slot, and executes it if it exists. The local reference frame of an entity is redefined in the inertial reference frame based on the position and orientation of the entity.

Constraints of the input specify required ranges of property values, relations between property values, extreme values which optimize some criteria, or any specific methods to solve the problem. The constraints are first handled by Algorithm 3.4.

Variables of the problem statement become the goals whose values are being sought during the problem solving process. Multiple variables can be specified in the input and they are computed simultaneously. Algorithm 3.5 first identifies the types of the variables by consulting the corresponding slots of their frame structures. If the value of a variable is already known or derivable from other known values, it is immediately obtained. If the value of a variable can be computed from other variables of the input, the equation describing their relations is derived and put in a set DRVD. If a variable is not dependent on any other variables, a new set of Maple variable names are created, which will be used in equations later. For example, if Oracle is asked to compute the position of a block b1, it generates a set of new variables \{b1x(t), b1y(t), b1z(t)\} for the position vector by concatenating the entity name b1 and the components [x(t), y(t), z(t)] of the position variable, which appear in the form facet of the position
Algorithm 3.2 ORACLE’s top-level algorithm

Problem Analysis Analyze a problem statement.

1. Analyze entities, and create frames of the entities and a set INIT of initial-value conditions using Algorithm 3.3.
2. Analyze constraints using Algorithm 3.4.
3. Analyze variables and generate a set DRVD of differential equations using Algorithm 3.5.

Model Creation Search for relevant model fragments and compose a behavioral model with them.

1. Using Algorithm 3.6, retrieve relevant model fragments, and put them in a list MFS.
2. model $M = DRVD$
3. $#equations = #equations(M)$
4. retry: For each model fragment $MF$ in MFS
   (a) Instantiate a model fragment $MF$ in MFS for the given problem using Algorithm 3.7.
   (b) $M = M \cup \{MF\}$
   (c) $#equations = #equations(M)$
   (d) If $#equations = #variables$, do Model Execution.
5. Print the dead-end situation, and quit.

Model Execution Solve the model $M$ either analytically or by numeric simulation.

1. Determine the types of equations of the model and attempt to solve them using Algorithm 3.8.
2. If a valid solution is obtained, print the model and solution, and quit.
3. If a valid solutions is not obtained, retract the most recent $MF$ from the model and go to retry.
Algorithm 3.3 Analyze entities of the input

If entity $E$ is an instance of an entity class currently supported by Oracle (see Figure B.1 for the taxonomy of entities)

1. Copy a frame of the entity class and let the frame be $F$.
2. $F[\text{name}] = \text{entity name}$
3. Change AKO slot to ISA slot and set the ISA slot to the entity class.
4. For each property value $PV$ specified in the problem statement
   - Set the value of the property slot of the frame $F$ to $PV$.
   - If there is an if-added facet in the slot
     - Execute the procedure attached to the facet.
5. For each initial value $IV$ specified in the problem statement
   - Put it in a set INIT.
6. For each vector $v'$ expressed in the local reference frame of the entity
   - Let entity position vector $r = (x, y, z)$
   - Entity orientation vector $q = (\phi, \theta, \psi)$
   - Then rotation matrix $S =
     \begin{pmatrix}
     \cos \psi \cos \phi - \cos \theta \sin \phi \sin \psi & -\sin \psi \cos \phi - \cos \theta \sin \phi \cos \psi & \sin \theta \sin \phi \\
     \cos \psi \sin \phi + \cos \theta \cos \phi \sin \psi & -\sin \psi \sin \phi + \cos \theta \cos \phi \cos \psi & -\sin \theta \cos \phi \\
     \sin \theta \sin \psi & \sin \theta \cos \psi & \cos \theta
     \end{pmatrix}
   - Transform $v'$ into a vector $v$ in the inertial reference frame by $v = r + S \times v'$

Else

1. OUTPUT ("entity $E$ is not yet implemented")
Algorithm 3.4 Analyze constraint of the input

For each constraint $C$
   If $C$ contains ‘<’, ‘>’, ‘<=’, or ‘>=’
      $C$ is of type range.
   Else if $C$ contains ‘=’
      $C$ is of type equality.
   Else if $C$ contains ‘maximize’ or ‘minimize’
      $C$ is of type optimize.
   Else if $C$ contains ‘Subs’, ‘SolveBy’, ‘AddEqu’, or ‘DropEqu’
      $C$ is of type function.
   Else
      OUTPUT ("$C$ is not supported")

Put each constraint $C$ in a list of its type.

slot of the frame of $b1$.

Potentially relevant model fragments are indexed by “mf” slot of each entity, starting with the most frequently relevant one first. Algorithm 3.6 searches for the model fragments and retrieves ones which will be instantiated by Algorithm 3.7 for the problem.

Algorithm 3.7 instantiates the retrieved model fragments by substituting variable names and values into the equations of the model fragment for the given problem. A set of equations from instantiated model fragments is solved using Algorithm 3.8.
Algorithm 3.5 Analyze variables of the input

For each variable $V$
- Determine the type (vector/scalar, time-independent/time-dependent, basic/derivable)
- If the value of $V$ is known (i.e., the value facet of the property slot has non-NULL value)
  - Retrieve it and adjust the variable list of the input.
- Else if the value of $V$ is derivable from other known quantities
  - Compute it using the if_needed procedure and adjust the variable list of the input.
- Else if $V$ is derivable from other variables specified in the input
  - Derive the equation for the relation using the if_needed procedure and put it in a set DRVD.
- Else
  - Generate a new set of Maple variable names by concatenating the entity name and the components of the variable.

Algorithm 3.6 Retrieve relevant model fragments

For each entity $E$ of the input
- For each model fragment $MF$ in the “mf” slot of $E$
  - If $MF$ has not been instantiated for $E$
    - AND
      - If every variable of $MF$ either corresponds to an entity property or variable mentioned in the input or can be derived from them
      - AND
        - If the assumption (if any) of $MF$ does not violate any entity property or constraint of the input
          - Put $MF$ in a list MFS.
Algorithm 3.7 Instantiate a model fragment

1. Replace the entity and variable names of the model fragment $MF$ by those of the input.
2. For each variable $V$ of $MF$
   Retrieve the corresponding slot from the entity frame.
   If the value of $V$ is known (i.e., 'value' facet has non-NUL1 value)
   Substitute the value into the equations of $MF$.
   Else if the function of $V$ appears in DRVD
   Substitute the function into the equations of $MF$.
   Else if there is an if needed procedure in the slot of the entity and
   the function of $V$ is derivable from other known values
   Call the procedure and substitute the returned value into the
   equations of $MF$.
   Else
   Generate a new Maple variable name for the variable and
   substitute the name into the equations of $MF$.
3. If the equations of $MF$ are differential equations and they are not expressed in canonical forms, convert them into canonical forms (see section 3.1.4 for canonical forms of differential equations).
Algorithm 3.8 Solve the equations of a model

If the equations are ordinary differential equations
  If there is no solving method specified by a constraint of the input
    Solve the equations with the initial-value conditions of INIT
    analytically using a Maple function “dsolve”.
    If Maple does not find a solution in closed form
      Solve the equations with the initial-value conditions of INIT using
      the Runge-Kutta method in section 3.1.4 and save the numeric
      solution in files.
  Else if there is a solving method specified by a constraint of the input
    Solve the equations with the initial-value conditions of INIT using
    that method and save the numeric solution in files.
Else if the equations are algebraic
  If there is no solving method specified by a constraint of the input
    Solve the equations algebraically using a Maple function “solve”.
    If Maple does not find a solution
      return error to the top-level algorithm.
  Else if there is a solving method specified by a constraint of the input
    Solve the equations using the method specified and return the result.
3.4 Examples

All the algorithms described in previous section have been implemented in a working program and tested on several types of mechanical systems. This section illustrates how the program works with a few examples.

3.4.1 Single Spring-Block System

Figure 3.7 reproduces the single spring-block system introduced in Chapter 1. Suppose that the problem statement in Figure 3.8 is given as an input for the problem of predicting the behavior of the single spring-block system (see Appendix B for the block and spring entity). b1 is a block with mass 1 and with principal moments of inertia [1/6,1/6,1/6]. The local reference frame of b1 is assumed to be fixed to b1 with its origin at the center of mass of b1. The local reference frame is initially located at [3,0,0] and oriented [\pi/4,\pi/2,0] without moving with respect to the inertial reference frame. s1 is a spring with force constant 10 and with damping coefficient 1/10. One end of s1 is attached to a fixed point, in this case the origin of the fixed inertial reference frame, but the other end is attached to a point [-1/2,0,0] of the block b1, represented in the local reference frame of b1. sb is a composite object with the block b1 and the spring s1 as its parts. There is no particular constraint in this problem and the system is asked to compute the four vector variables (twelve variables in component form)
entities=[b1=[block, mass=1,  
    principal_moments_of_inertia=[1/6, 1/6, 1/6],  
    position(0)=[3,0,0], orientation(0)=[Pi/4, Pi/2, 0],  
    velocity(0)=[0,0,0], ang_velocity(0)=[0,0,0]],  
  s1=[spring, force_const=10,  
    damping_coeff=1/10, rest_length=3/2,  
    end1(t)=[0,0,0], end2(t)=b1[-1/2, 0, 0]],  
  sb=[composite_object, parts={b1, s1}]];  
constraints=[ ];  
variables=[b1[position(t)], b1[orientation(t)],  
  b1[velocity(t)], b1[ang_velocity(t)]];  

Figure 3.8: Instance of the problem statement data structure for the single spring-block system

of the block.

For each entity b1, s1, and sb, a frame structure is created and the given properties of the entities are recorded in their slot values using Algorithm 3.3. The position of end2 in the inertial reference frame is computed from a translation and a rotation of the local reference frame of the block b1 using Algorithm 3.3. Since there is an if_added procedure in the end2 slot of a spring, it is also executed. The procedure instantiates a model fragment of Springforce2 and records the value of the force acting on the block in its force slot. For each of the initial-value conditions of b1 specified in the problem statement, the form facet in the corresponding slot of the frame of b1 is consulted to formulate initial conditions with Maple variable names. The form facets of the position, orientation, velocity, and ang_velocity slots contain [x(t), y(t), z(t)], [phi(t), theta(t), psi(t)], [u(t), v(t), w(t)], and [omega1(t), omega2(t), omega3(t)], respectively. Therefore, Oracle assigns Maple variable names by concatenating the entity name b1 and the components of each form facet, and formulates initial conditions with the names.

INIT = {b1x(0)=3, b1y(0)=0, b1z(0)=0,
b1phi(0)=0, b1theta(0)=Pi/2, b1psi(0)=0,
b1u(0)=0, b1v(0)=0, b1w(0)=0,
b1omega1(0)=0, b1omega2(0)=0, b1omega3(0)=0}

None of the four state variables of the block can be assigned a value simply by looking at slot values of b1, but the velocity and the angular velocity functions can be derived by differentiating the position and the orientation functions, respectively, according to their if_needed facets. Figure 3.9 and 3.10 show Maple procedures attached to the if_needed facets for deriving velocity and angular velocity, respectively. Algorithm 3.5 generates trivial differential equations for the velocity and angular velocity by calling the procedures in Figure 3.9 and 3.10.

\[
\text{DRVD} = \{ b1u(t) = \text{diff}(b1x(t), t), \\
            b1v(t) = \text{diff}(b1y(t), t), \\
            b1w(t) = \text{diff}(b1z(t), t), \\
            b1omega1(t) = \text{diff}(b1theta(t), t) * \cos(b1phi(t)) + \\
                          \text{diff}(b1psi(t), t) * \sin(b1theta(t)) * \sin(b1phi(t)), \\
            b1omega2(t) = \text{diff}(b1theta(t), t) * \sin(b1phi(t)) - \\
                          \text{diff}(b1psi(t), t) * \sin(b1theta(t)) * \cos(b1phi(t)), \\
            b1omega3(t) = \text{diff}(b1phi(t), t) + \text{diff}(b1psi(t), t) * \cos(b1theta(t)) \}
\]

Now Oracle focuses on finding equations for the position and the orientation. The equations for them cannot be derived from other variables since they are basic variables, so Oracle looks for relevant model fragments using Algorithm 3.6. It examines model fragments, indexed by the mf slot of the block. Oracle decides that Newton2 and Euler are potentially relevant because the entities (solid and rigid body, respectively) of the model fragments are super classes of a block and the equations of the model fragments contain at least one variable of the problem. Model fragments for Newton2 and Euler are as follows. ²

²Syntax is slightly modified for readability. See Appendix A for the actual representation of the model fragments in Maple.
derive_vel := proc(pos)
local c, newc, facet, slot, frame, vname, fvalue, i:

vname := op(0, pos):
for i from 1 to nops(entities) do
  if ename.i=vname then
    frame := e.i:
    # Find a frame structure representing the i-th entity.
  fi:
od:

slot := frame[op(1, pos)]: # extract the slot name of the parameter.

for facet in slot do
  if lhs(facet) = 'value' then
    fvalue := rhs(facet):
    if not (fvalue=null) then
      RETURN(map(diff, fvalue, t)):
      # The map function applies "diff" to each component
      # of fvalue, and each component of fvalue is replaced
      # by its time derivative.
    fi:
  elif lhs(facet) = 'form' then
    fvalue := eval(rhs(facet)):
    for i from 1 to 3 do
      c := op(0, fvalue[i]):
      newc := cat(vname, c):
      fvalue := subs(c=newc, eval(fvalue)):
      # Every occurrence of an expression represented by c
      # in fvalue is substituted with an expression
      # represented by newc by the subs function.
    od:
    RETURN(map(diff, fvalue, t)):
    # Each component of fvalue is replaced by its
    # time derivative by the map function.
  fi:
od:
ERROR('cannot derive velocity'):
end:

Figure 3.9: Maple procedure attached to the if_needed facet of the velocity slot
derive_ang_vel:= proc(ortnt)
local c, newc, facet, slot, frame, vname, fvalue, i:

vname := op(0, ortnt):
for i from 1 to nops(entities) do
  if ename.i=vname then
    frame := e.i: # Find a frame representing the i-th entity.
  fi:
od:
slot := frame[op(1, ortnt)]: # extract the slot name of the parameter.
for facet in slot do
  if lhs(facet) = 'value' then
    fvalue := rhs(facet):
    if not (fvalue=null) then
      RETURN(linalg[vector](3, [diff(fvalue[3],t)*sin(fvalue[2])*sin(fvalue[1])+diff(fvalue[2],t)*cos(fvalue[1]),
                                  -diff(fvalue[3],t)*sin(fvalue[2])*cos(fvalue[1])+diff(fvalue[2],t)*sin(fvalue[1]),
                                 diff(fvalue[3],t)*cos(fvalue[2])+diff(fvalue[1],t)])):
# The function linalg[vector](n, [a,b,c]) produces a vector
# of length n containing the elements a, b, and c.
    fi:
  elif lhs(facet) = 'form' then
    fvalue := eval(rhs(facet)):
    for i from 1 to 3 do
      c := op(0, fvalue[i]):
      newc := cat(vname, c):
      fvalue := subs(c=newc, eval(fvalue)):
    od:
    RETURN(linalg[vector](3, [diff(fvalue[3],t)*sin(fvalue[2])*sin(fvalue[1])+diff(fvalue[2],t)*cos(fvalue[1]),
                                 -diff(fvalue[3],t)*sin(fvalue[2])*cos(fvalue[1])+diff(fvalue[2],t)*sin(fvalue[1]),
                                diff(fvalue[3],t)*cos(fvalue[2])+diff(fvalue[1],t)])):
# A vector with 3 components is created by the function
# linalg[vector].
  fi:
od:
ERROR('cannot derive ang_velocity'):
end:

Figure 3.10: Maple procedure attached to the if_needed facet of the ang_velocity slot
Newton2={phenomenon='Newton’s second law of motion',
    entities=[r=solid],
    variables=[f(t)=r[net_force(t)], p(t)=r[momentum(t)]],
    assumptions=[ ],
    equations=[f(t)=diff(p(t),t)]

Euler={phenomenon='the time-dependency of the angular velocity',
    entities=[b=rigid_body],
    variables=[Omega(t)=b[ang_velocity(t)],
               M(t)=b[ang_momentum(t)],
               T(t)=b[net_torque(t)]],
    assumptions=[ ],
    equations=[add(diff(M(t),t), crossprod(Omega(t),M(t))) = T(t)]

Note that all the variables f(t), p(t), Omega(t), M(t), and T(t) in the above model fragments are vectors. The entity and variable names of the model fragments are instantiated as those of the problem and they are substituted in the equations of the model fragments using Algorithm 3.7. The angular momentum is derived from principal moments of inertia and angular velocity by the if_needed procedure in the ang_momentum slot. Likewise, the net torque is derived from force and position vector of the point at which the force acts.

\[
M(t) = \begin{pmatrix} I_1 \Omega_1(t) \\
               I_2 \Omega_2(t) \\
               I_3 \Omega_3(t) \end{pmatrix}, \quad T(t) = \sum r \times f(t)
\]

The position vector \( r \) of the spring-attached point and the principal moments of inertia \( (I_1, I_2, I_3) \) can be assigned from the information of the problem description. The angular velocity is one of the state variables asked by the problem, and its functions are derived in DRVD. The force \( f(t) \) has been already computed using a model fragment of Springforce2 (shown earlier in Section 2.3).
during the analysis of entities. Therefore, the force value available in the force
slot of b1 is substituted in the equations of the model fragments Newton2 and
Euler. The system has now a model (see appendix C for the model generated by
\textsc{oracle}) described by 12 differential equations in component form (6 from the
model fragments and 6 from DRVD) plus 12 initial conditions for 12 unknowns
(3 components from each of the 4 vector variables).

Several of the differential equations are nonlinear, and when \textsc{oracle} attempts
to solve the model analytically, it does not find a solution in closed form. \textsc{oracle}
then solves the differential equations by numeric simulation using Algorithm 3.8
which calls Algorithm 3.1 by default. \textsc{oracle} displays the simulation result by
showing the state variables as functions of time using \texttt{gnuplot} \footnote{A free program which is available via internet anonymous ftp.}(Figure 3.12). An-
imation of the moving block is shown (Figure 3.11) using PADL-2 solid modeling
system [Hartquist, 1983]. Figure 3.11 contains several animation scenes super-
imposed. Note that the motion of the block is much more complex than that of
standard one-dimensional spring-block system (i.e., linear harmonic oscillator).
The kinetic energy, potential energy, and total energy of the problem are also dis-
played as part of validation criteria of the results (Figure 3.13). The total energy
in Figure 3.13 decreases over time due to the nonzero damping coefficient of the
spring of the problem statement. The total CPU time for generating the model
and simulating it with 1000 time steps on a Sun SparcStation is 1225.85 seconds
(20 minutes 25.85 seconds). Detailed information on the time for generating the
model and solving it appears later in Table 3.1 of Section 3.6.

When the model is transformed into a model in the C programming language,
it can be solved in much less time. The way to generate a model in Maple is
effectively the same as before. \textsc{oracle} then transforms the differential equations of
the model into a C subroutine called “\texttt{derivs}”, which returns derivatives $dy/dt$ at $t$. 

\footnote{A free program which is available via internet anonymous ftp.}
The initial conditions of the differential equations are put into a C main routine with a statement calling a driver routine called "rkdumb". ORACLE compiles the main routine and the subroutine derivs with the publicly-available archive library containing the definitions of the subroutines rkdumb and rk4, which actually implements the fourth-order Runge-Kutta method (shown in Algorithm 3.1). It then executes the resulting code to perform numeric simulation. During the simulation, the driver routine rkdumb calls the subroutine rk4 and stores the results of the numeric integration. The result of the simulation by the compiled code is as accurate as that by a Maple procedure implementing Algorithm 3.1 when the same step size is used. However, it is much faster; the total CPU time for generating the model and simulating it with 1000 time steps on a Sun SparcStation is 49.22 seconds. Times for generating models and solving them for this example and other examples are summarized later in Table 3.1 of Section 3.6.
Figure 3.12: Plots of the 12 state variables of the block b1 of the single spring-block system as functions of time
Figure 3.13: The kinetic, potential, and total energy of the single spring-block system as a function of time during the simulation
Figure 3.14: 2 blocks connected by 1 spring. Both blocks are pulled in opposite directions and released.

Figure 3.15: 3 blocks connected by 2 springs. The middle block is pulled directly to the side and released.

3.4.2 Multiple Spring-Block System

The previous section showed how ORACLE predicts the behavior of the single spring-block system. Can the modeling system of the single spring-block system be used to predict the behavior of the multiple spring-block systems such as Figure 3.14 and Figure 3.15? The answer is “yes”. The multiple spring-block systems are analogous to the single spring-block system; they have additional entities and phenomena, but they are simply the multiple occurrences of the same types as the single spring-block system. Having already enough knowledge represented in general form to handle the single spring-block system, ORACLE can handle the multiple spring-block systems with no change. In fact ORACLE is able to handle multiple rigid bodies connected by springs in arbitrary positions and orientations because the way of identifying relevant model fragments and
entities := [b1=[block, mass=1, principal_moments_of_inertia=[1/6,1/6,1/6], position(0)=[-5/2,-1/2,0], orientation(0)=[0,pi/2,0], velocity(0)=[0,0,0], ang_velocity(0)=[0,0,0]],
b2=[block, mass=1, principal_moments_of_inertia=[1/6,1/6,1/6], position(0)=[5/2,1/2,0], orientation(0)=[0,pi/2,0], velocity(0)=[0,0,0], ang_velocity(0)=[0,0,0]],
s1=[spring, force_const=10, damping_coeff=1/10, rest_length=3, end1(t)=b1[1/2,0,0], end2(t)=b2[-1/2,0,0]],
sb=[composite_object, parts={b1,b2,s1}];
constraints := [];
variables := [b1[position(t)], b1[orientation(t)], b1[velocity(t)], b1[ang_velocity(t)], b2[position(t)], b2[orientation(t)], b2[velocity(t)], b2[ang_velocity(t)]];

Figure 3.16: Input to the two-block system

composing them is not restricted by the number of entities or their connections.

Consider the problem statement in Figure 3.16 given as input for the two-block system. The input for the three-block system is given in a similar problem statement.

The way ORACLE solves the problem for multiple spring-block systems is the same as that for the single spring-block system. It computes the positions of ends of each spring in the inertial reference frame by transforming the local reference frame of its associated block by Algorithm 3.3, and spring forces acting on the blocks using model fragments of SpringForce1 and SpringForce2. It derives differential equations for the velocity and angular velocity of the blocks from the
Figure 3.17: Motion of the two-block system with a spring of nonzero damping coefficient. Spring not shown.

procedures attached to the if-needed facets using Algorithm 3.5. It then retrieves and instantiates model fragments of Newton2 and Euler using Algorithm 3.6 and Algorithm 3.7, and composes a model. Notice that model fragment sharing occurs within the models because each of those model fragments is instantiated more than once for different entities. The result of the execution of the models indicate that although none of the blocks are initially rotated, both blocks of Figure 3.14 and the end blocks of Figure 3.15 rotate as well as translate due to spring forces which are not parallel to the radius vectors of the points to which the springs are attached (see Figures 3.17, 3.18, and 3.19). If the spring damping is ignored (i.e., damping_coeff = 0), the middle block of Figure 3.15 shows translational motion only (Figure 3.18 and 3.20), but it shows both translational and rotational motions if the spring damping is considered (Figure 3.19 and 3.22). The times for generating the models and solving them are summarized later in Table 3.1.
Figure 3.18: Motion of the three-block system with springs of zero damping coefficients

Figure 3.19: Motion of the three-block system with springs of nonzero damping coefficients
Figure 3.20: Plots of the 12 state variables of the middle block of the three-block system with springs of zero damping coefficients as functions of time
Figure 3.21: The kinetic, potential, and total energy of the three-block system as a function of time during the simulation for the case with zero damping coefficient.
Figure 3.22: Plots of the 12 state variables of the middle block of the three-block system with springs of nonzero damping coefficients as functions of time.
Figure 3.23: The kinetic, potential, and total energy of the three-block system as a function of time during the simulation for the case with nonzero damping coefficient.
3.4.3 Sailboat in Fluid

A sailboat (see Figure 3.24 and 3.25) is a composite object whose driving force comes from the differential motion of air over water. Before we model the sailboat, we can ask the same question as before: can we use the modeling system of the spring-block systems to predict the behavior of a sailboat in fluids? The answer is “yes” provided that the modeling system has enough domain knowledge to handle the problem. We do not need to build a different modeling system. A modeling system with the same algorithm and the same model fragments as before plus additional model fragments and entities can predict the behavior of the sailboat.

New classes of entities added to the knowledge base are fluids (water and air) and lifting surfaces (hull and sail). See Appendix B for the representation of the entities. A sailboat is a composite object with hull and sail as its parts. The sailboat, water, and air entity have their own reference frames, which move as their entities move. The reference frame of water will be used as the inertial reference frame for the sailboat problem. When a boat sails in light winds, tilting of the boat about the longitudinal axis is small and does not affect the performance of the sailboat. Thus, two Euler angles $\theta$ and $\psi$ can be regarded as zero, and

Figure 3.24: *Stars & Stripes*, winner of the 1987 America’s Cup competition
Figure 3.25: Directions of force components on a sailboat, adapted from [Letcher, 1976]

the orientation of an entity is conveniently represented with only one parameter, \( \phi \). We call the angle \( \phi \) the direction of the entity. The representation of the orientation by the direction \( \phi \) can be converted to the representation by the Euler angles \((\phi, 0, 0)\), and vice versa, by if_needed procedures. As mentioned earlier, the direction is measured with respect to the water reference frame.

New phenomena include hydrodynamic forces and aerodynamic forces, each with two components (lift and drag), and skin friction. A single model fragment is used to represent both hydrodynamic and aerodynamic frictional drag forces, and later instantiated for them. Likewise, a single model is used to represent both hydrodynamic and aerodynamic lift forces.

\[
F_{\text{Drag}} = \text{[phenomenon='frictional drag force on an object in fluid', entities=[s=physical_object, f=fluid], variables=[FD=s[fdrag(t)], v=s[rel\_fluid\_speed(t)], fd=s[rel\_fluid\_direction(t)], Pa=s[parasitic\_area], rho=f[density]]}},
\]

```
assumptions=[ ],
equations=[FD=1/2*Pa*rho*v^2*fd]]

Lift=[phenomenon='lift and lift induced forces on an object in fluid',
entities=[s=physical_object, f=fluid],
variables=[LF=s[lift(t)],
L=s[lift_magnitude(t)],
v=s[rel_fluid_speed(t)],
fd=s[rel_fluid_direction(t)],
pd=s[perpendicular_rel_fluid_dir(t)],
Ca=s[effective_capture_area],
rho=f[density]],
assumptions=[s[rel_fluid_speed(t)] > 0],
equations=[LF=L*pd+L^2/(2*Ca*rho*v^2)*fd]]

For the hull, the rel_fluid_speed is the speed of the boat relative to water. For the sail, the rel_fluid_speed is the speed of the boat relative to air. The magnitude and direction of the hydrodynamic forces acting on the hull depend on the rel_fluid_speed of the hull, and the aerodynamic forces depend on the rel_fluid_speed of the sail. rel_fluid_direction is an angle between the direction of fluid and the direction of an object, represented in angle. Drag forces acting on the hull have components opposite to the direction of its rel_fluid_speed. Lift forces on the hull are perpendicular to the direction of its rel_fluid_speed. Similarly, drag forces on the sail have components opposite to the direction of its rel_fluid_speed, and lift forces on the sail are perpendicular to the direction of its rel_fluid_speed. Lift forces on the hull and sail of a sailboat are horizontal, not vertical. The directions of the force components are summarized in Figure 3.25. In the figure, water is assumed to be at rest (i.e., speed = 0) and $V_t$, $V_s$, and $A$ denote the wind speed, sailboat speed, and course angle from the wind direction, respectively.
Notice that the model fragment Lift has a nonempty assumption slot, saying that the relative fluid speed must be positive. When both the fluid and the object are at rest or the fluid has the same speed as the object in the same direction, the relative fluid speed becomes zero and the equations of the model fragment cannot be defined due to zero denominator. Thus we have another model fragment of lift with a different assumption slot; it says that lift force is zero when the relative fluid speed is zero.

\[
\text{Lift\_at\_zero\_speed=}
\]

\[
\text{phenomenon='lift and lift induced force on an object in fluid',}
\]

\[
\text{entities=[s=physical\_object, f=fluid],}
\]

\[
\text{variables=[L=s[lift(t)],}
\]

\[
\text{sv=s[rel\_fluid\_speed(t)]],}
\]

\[
\text{assumptions=[s[rel\_fluid\_speed(t)] = 0],}
\]

\[
\text{equations=[L[1]=0,}
\]

\[
\text{L[2]=0,}
\]

\[
\text{L[3]=0]}
\]

During problem solving, ORACLE automatically chooses between the two model fragments of lift by checking their assumptions. At present, the kinds of assumptions ORACLE can process are confined to the algebraic properties of entities, such as numeric ranges or arithmetic expressions. After the entities and model fragments are added, ORACLE can solve several types of problems on a sailboat such as

- Given the wind velocity (or equivalently, the wind speed and direction) and the boat direction, find the boat speed which will balance all the forces involved.

- Given the wind velocity and the boat direction, find the optimum boat speed with the maximum component in the wind direction.
entities := [w=[water, direction(t)=0, speed(t)=0],
    a=[air, direction(t)=.86, speed(t)=16.9],
    h=[hull, fluid=w, effective_capture_area=24.7,
      parasitic_area=0.577],
    s=[sail, fluid=a, effective_capture_area=896,
      parasitic_area=24],
    b=[sailboat, parts={h,s}, direction(t)=0,
      net_force(t)=[0,0,0]]];

constraints := [];
variables := [b[speed(t)]];
equation of Newton2, \( F(t) = \frac{d(p(t))}{dt} \). Since the problem states that all the forces are balanced, the net force on the sailboat must be zero, implying the momentum \( p(t) \) is constant. The right hand side of the equation becomes zero from the constant momentum, resulting in an algebraic equation. However, the problem is under-constrained in the sense that total number of equations in component form is 2 (\( F_x = 0 \), \( F_y = 0 \), \( F_z \) becomes a trivial equation \( 0 = 0 \)) but the total number of unknowns in the equations is 3 (boat speed, sail lift magnitude, and hull lift magnitude). Oracle prints the situation, asking for further information. The user provides an additional equation as a constraint, \( \frac{\partial(F_x)}{\partial(sail\_lift\_mag)} = 0 \), in a new problem statement by making a simplifying assumption that the parts of the sailboat are controlled as to maximize the sailboat force in the direction of boat heading (Figure 3.27). The other two constraints in the problem statement guide the way of solving the equations: solve the equation of \( F_y \) (i.e., net force in the \( y \) direction) for \( lift\_magnitude \) and substitute the result in the equation of \( F_x \), and solve the equations using floating-point arithmetic (Maple “fsolve” function). Oracle formulates a model of algebraic equations (Figure 3.31), solves the equations of the model with the user-supplied equation using the solving method specified in the constraints. In this problem a single solution is obtained for the boat speed, 15.8 ft/sec, and the solution is checked against the range facet of the speed slot of the boat, which says that value of the boat speed must be in the range of \([0 .. \infty]\). Since the solution of 15.8 is included in the range, it is returned to the user as a valid solution. However, if there are several solutions, only those within the values specified by the range facet (if any) are selected as valid solutions. When the information in the range facet is not sufficient to choose a correct solution from multiple solutions, additional or alternative procedure for filtering correct solutions is to use the result of the numeric simulation of its corresponding differential model, as we will show below. A correct solution of an algebraic model describing a behavior in a steady state must agree with the
entities := [w=[water, direction(t)=0, speed(t)=0],
a=[air, direction(t)=-.86, speed(t)=16.9],
h=[hull, fluid=w, effective_capture_area=24.7,
   parasitic_area=0.577],
s=[sail, fluid=a, effective_capture_area=896,
   parasitic_area=24],
b=[sailboat, parts={h,s}, direction(t)=0,
   net_force(t)=[0,0,0]];
constraints := [AddEqn(Diff(b[net_force(t)][1], s[lift_mag(t)])=0),
   Subs(h[lift_mag(t)]=Solve(b[net_force(t)][2],
      h[lift_mag(t)]), b[net_force(t)][1]),
   SolveBy(fsolve)];
variables := [b[speed(t)]];

Figure 3.27: Input for the algebraic model of the sailboat problem

numeric simulation result of its corresponding differential model. However, this
procedure of validating the solution of an algebraic model against the numeric
simulation of a differential model has not been automated in the current imple-
mentation of ORACLE, and the user should try both models to compare their
results.

The previous example showed how ORACLE composes a model to compute
the sailboat speed in the equilibrium state of forces. If we are interested not
only in such a speed but also in how the boat arrives at the speed, starting from
zero speed, the boat speed must be computed as a function of time. The input
to this problem appears in Figure 3.28. Note that there is no statement about
the net force of the boat in the problem statement. Relevant model fragments
are retrieved and instantiated in a similar way. In this case, however, the net
force on the sailboat is not necessarily zero all the time because the boat accel-

The right hand side of the equation of Newton2 does not become zero, but stays as d(p(t))/dt. ORACLE substitutes m \cdot v(t) for p(t), and forms a model (see Figure 3.32). The
entities := [w=[water, direction(t)=0, speed(t)=0],
    a=[air, direction(t)=.86, speed(t)=16.9],
    h=[hull, fluid=w, effective_capture_area=24.7,
        parasitic_area=0.577],
    s=[sail, fluid=a, effective_capture_area=896,
        parasitic_area=24],
    b=[sailboat, parts={h,s}, direction(t)=0,
        mass=100, speed(0)=0]];

constraints := [AddEqn(Diff(b[net_force(t)])[1], s[lift_mag(t)])=0),
    Subs(h[lift_mag(t)]=Solve(b[net_force(t)])[2],
        h[lift_mag(t)], b[net_force(t)])[1]];

variables := [b[speed(t)]];

differential equation, \( F(t) = d(m \cdot v(t))/dt \), in the model is solved for \( v(t) \) by numeric simulation. At each step of the numeric simulation, Oracle evaluates \( F \) by solving \( \partial(Fx)/\partial(sail\_lift\_mag) = 0 \) and substitutes the result of solving \( Fy \) for \( lift\_magnitude \) in the equation of \( Fx \) to find the lift force maximizing \( F \) for the current boat speed. A plot of the simulation result in Figure 3.29 shows that the sailboat ultimately accelerates to the same speed as the one predicted by the algebraic method, thus confirming the algebraic solution. Also notice that the model fragment of Newton2 used for modeling the spring-block systems is reused for modeling the sailboat and that model fragments Lift and FDrag are shared by hull and sail.

A sailboat can be directed along a wide range of course angles \( A \) with respect to wind. In each of these directions, there is an attainable boat speed \( V_s \), which will balance all the forces acting on the boat. Figure 3.30 shows these speeds represented in the form of a \textit{speed polar} [Letcher, 1991], where \( A \) is the course angle \( (0 \leq A \leq \pi \text{ in the figure}) \) and \( V_s \) is the radius. The same wind speed and boat parameters (except the direction of the boat) as those shown earlier in a query are used to produce the speed polar. In the figure, the negative x-axis is
Figure 3.29: The sailboat speed as a function of time

the direction of wind and the course angle is measured from the positive x-axis (i.e., upwind direction). The speed polar for the course angle $\pi \leq A \leq 2\pi$ is symmetric with Figure 3.30 with respect to the x-axis. Notice that for the given wind speed 16.9 ft/sec the maximum boat speed 24.1 ft/sec (marked by an arrow) is attained at the downwind course angle $A \approx 100$ degrees, which is different from 180 degrees. Table 3.1 in Section 3.6 summarizes the times for generating and solving the algebraic and differential models of the sailboat problem.
Figure 3.30: A speed polar showing the boat speed $V_s$ as a function of the course angle $A$ for the wind speed 16.9 ft/sec. The negative x-axis is the wind direction and the positive x-axis is the upwind direction. The course angle is measured from the upwind direction. The radius from the origin is the boat speed.
model = [
  variable={bspeed(t),

  assumption={0 < s[rel_fluid_speed(t)], 0 < h[rel_fluid_speed(t)],

  model_fragments={newton2, FDrag, Lift},

  equation=[0 = -46895.0E-8*slift_mag(t)*(-77997.0E2-60226.0E1*
          bspeed(t)-27311.*bspeed(t)^2+5513.*slift_mag(t)+500.*slift_mag(t)*
          bspeed(t))/(285.59+22.052*bspeed(t)+bspeed(t)^2)^2/(3/2)-57120.0E-9*
          (285.59+22.052*bspeed(t)+bspeed(t)^2)^2/(1/2)*(5513.+500.*bspeed(t))-
          10208.0E-6*(3149.1*slift_mag(t)/(285.59+22.052*bspeed(t)+bspeed(t)^2)^2*
          (3/2)+528.78*slift_mag(t)/(285.59+22.052*bspeed(t)+bspeed(t)^2)^2/(3/2)+
          3.6576*(285.59+22.052*bspeed(t)+bspeed(t)^2)^2/(1/2)))/2*bspeed
          (t)^2-0.57210*bspeed(t)^2, 0 = -46895.0E-8*(-77997.0E2-60226.0E1*bspeed(t)-27311.*bspeed(t)^2+
          5513.*slift_mag(t)+500.*slift_mag(t)*bspeed(t))/(285.59+22.052*
          bspeed(t)+bspeed(t)^2)^2-(3/2)-46895.0E-8*slift_mag(t)*(-5513.+500.*
          bspeed(t))/(285.59+22.052*bspeed(t)+bspeed(t)^2)^2/(3/2)-20416.0E-6*
          (3149.1*slift_mag(t)/(285.59+22.052*bspeed(t)+bspeed(t)^2)^2/(3/2)+
          528.78*slift_mag(t)/(285.59+22.052*bspeed(t)+bspeed(t)^2)^2/(3/2)+
          3.6576*(285.59+22.052*bspeed(t)+
          bspeed(t)^2)^2/(1/2))/2*bspeed(t)^2-1.0001*slift_mag(t)/(285.59+22.052*bspeed(t)+
          bspeed(t)^2)^2/(3/2)+3.0030*slift_mag(t)^2/(285.59+22.052*
          bspeed(t)+bspeed(t)^2)^2/(3/2)+.36576*(285.59+22.052*bspeed(t)+
          bspeed(t)^2)^2/(1/2))/2*bspeed(t)^2+1.0001*slift_mag(t)/(285.59+22.052*
          bspeed(t)+bspeed(t)^2)^2/(3/2)+3.0030*slift_mag(t)^2/(285.59+22.052*
          bspeed(t)+bspeed(t)^2)^2/(3/2)+528.78*(285.59+22.052*bspeed(t)+
          bspeed(t)^2)^2/(3/2)+3.6576*(285.59+22.052*bspeed(t)+
          bspeed(t)^2)^2/(1/2))/2*bspeed(t)^2+1.0001*(285.59+22.052*bspeed(t)+
          bspeed(t)^2)^2/(3/2)+3.6060*slift_mag(t)/(285.59+22.052*bspeed(t)+
          bspeed(t)^2)^2/(3/2))]

Figure 3.31: Algebraic model formulated by ORACLE for the sailboat problem
(edited for readability)
model = [
variable={bspeed(t)},

model_fragment={newton2, FDrag, Lift, Lift_at_zero_speed},

assumption={0 < s[rel_fluid_speed(t)], 0 < h[rel_fluid_speed(t)],
  0 = h[rel_fluid_speed(t)]},

equation=[bspeed(0) = 0,
  diff(bspeed(t),t) = 46895.0E-10*slift_mag(t)*(77997.0E2+60226.0E1*
    bspeed(t)+27311.*bspeed(t)^2-5513.*slift_mag(t)-500.*slift_mag(t)*
    bspeed(t))/(285.59+22.052*bspeed(t)*bspeed(t)^2)^2(3/2)-57120.0E-11*
    (285.59+22.052*bspeed(t)*bspeed(t)^2)^2(1/2)*(5513.+500.*bspeed(t))-10208.0E-8*(3149.1*slift_mag(t)/(285.59+22.052*bspeed(t)*bspeed(t)^2)*
    (3/2)+528.78*slift_mag(t))/(285.59+22.052*bspeed(t)*bspeed(t)^2)^2(3/2)*bspeed(t)+33.080*slift_mag(t)/(285.59+22.052*bspeed(t)+
    bspeed(t)^2)^2(3/2)*bspeed(t)^2+1.0001*slift_mag(t)/(285.59+22.052* bspeed(t)+bspeed(t)^2)^2(3/2)*bspeed(t)^3+3.0030*slift_mag(t)^2/
    (285.59+22.052*bspeed(t)*bspeed(t)^2)^2(3/2)+36576*(285.59+22.052* 
    bspeed(t)+bspeed(t)^2)^2(1/2))/bspeed(t)^2-57210.0E-7*bspeed(t)^2, 
    0 = 46895.0E-10*(77997.0E2+60226.0E1*bspeed(t)+27311.*bspeed(t)^2- 
    5513.*slift_mag(t)-500.*slift_mag(t)*bspeed(t))/(285.59+22.052* 
    bspeed(t)+bspeed(t)^2)^2(3/2)+46895.0E-10*slift_mag(t)*(-5513.-500. * 
    bspeed(t))/(285.59+22.052*bspeed(t)+bspeed(t)^2)^2(3/2)-20416.0E-8* 
    (3149.1*slift_mag(t)/(285.59+22.052*bspeed(t)+bspeed(t)^2)^2(3/2)+
    528.78*slift_mag(t))/(285.59+22.052*bspeed(t)+bspeed(t)^2)^2(3/2)* 
    bspeed(t)+33.080*slift_mag(t)/(285.59+22.052*bspeed(t)+bspeed(t)^2)*
    (3/2)*bspeed(t)^2+1.0001*slift_mag(t)/(285.59+22.052*bspeed(t)+
    bspeed(t)^2)^2(3/2)*bspeed(t)^3+3.0030*slift_mag(t)^2/(285.59+ 
    22.052*bspeed(t)+bspeed(t)^2)^2(3/2)+36576*(285.59+22.052* 
    bspeed(t)+bspeed(t)^2)^2(1/2))/bspeed(t)^2*(3149.1/(285.59+22.052* 
    bspeed(t)+bspeed(t)^2)^2(3/2)+528.78/(285.59+22.052*bspeed(t)+ 
    bspeed(t)^2)^2(3/2)*bspeed(t)+33.080/(285.59+22.052*bspeed(t)+
    bspeed(t)^2)^2(3/2)*bspeed(t)^2+1.0001/(285.59+22.052*bspeed(t)+
    bspeed(t)^2)^2(3/2)*bspeed(t)+36.060*slift_mag(t)/(285.59+22.052* 
    bspeed(t)+bspeed(t)^2)^2(3/2)]]

Figure 3.32: Differential model formulated by ORACLE for the sailboat problem (edited for readability)
3.4.4 Disc Mounted on a Rotating Shaft

The examples shown so far have numeric values as their property values in the problem statements. When the property values are specified in terms of symbols, Oracle can still find a solution as long as a final set of equations to be solved are algebraic equations or differential equations which have a solution in closed form. Consider the following problem, taken from [Bradbury, 1968]. A uniform thin disc d1 of radius $a$ and mass $m$ is mounted on a shaft which turns at a constant angular velocity $\omega$. The shaft passes through the center of the disc, but the disc is tilted at an angle $\theta$ with respect to the shaft as pictured in Figure 3.33.

Given the input of Figure 3.34, the problem is to compute the net torque which the disc d1 must sustain. A body reference frame of the disc is assumed to have its origin at the center of the disc. The $y$-axis is perpendicular to the disc and is at an angle $\theta$ with respect to the shaft. The $x$-axis is perpendicular to the shaft. The components of the angular velocity of rotation is given with respect to the body reference frame of d1 as shown in Figure 3.34.

After a frame structure is created for each entity d1, r1, and dr, a new set of Maple variables is generated for the variable d1[net_torque(t)]: \{d1T1(t), d1T2(t), d1T3(t)\}. The value facet of the net torque slot in the frame structure of the disc
entities=[d1=[disc, mass=m, radius=a,
    principal_moments_of_inertia=
    [m*a^2/4, m*a^2/2, m*a^2/4],
    ang_velocity(t)=
    [0,omega*cos(theta),-omega*sin(theta)]],
    r1=[rod],
    dr=[composite_object, parts={d1,r1}];
constraints=[ ];
variables=[d1[net_torque(t)]];

Figure 3.34: Input for the disc-rod problem

d1 has NULL value (i.e., unknown so far). The if_needed procedure of the slot
says that the net torque can be derived from the force and the force position, but
their values are not available in this problem. Hence, Oracle searches for model
fragments which can be used to solve the problem using Algorithm 3.6, and finds
a model fragment Euler (see section 3.4.1 or appendix A for the model fragment).
The equations of Euler can compute the net torque from the angular velocity
and the angular momentum. The angular velocity of d1 is given in the problem
statement, and the angular momentum \( \mathbf{M}(t) \) of d1 can be computed from the
principal moments of inertia \( \mathbf{I} \) and the angular velocity \( \mathbf{\Omega}(t) \) of d1, both of which
are given in the problem statement.

\[
\mathbf{M}_1(t) = I_1*\Omega_1(t) = (m*a^2/4)*0 = 0
\]
\[
\mathbf{M}_2(t) = I_2*\Omega_2(t) = (m*a^2/2)*(\Omega_1*\cos(\theta)) = 1/2*m*a^2*\Omega_1*\cos(\theta)
\]
\[
\mathbf{M}_3(t) = I_3*\Omega_3(t) = (m*a^2/4)*(-\Omega_1*\sin(\theta)) = -1/4*m*a^2*\Omega_1*\sin(\theta)
\]

The angular momentum and angular velocity are substituted into the equations
of the model fragment of Euler using Algorithm 3.7. From the constant angular
velocity, \( \dot{\omega} = 0 \). Therefore, the equations of Euler become algebraic equations and they are solved by Algorithm 3.8 for the variables \( d1T_1(t) \), \( d1T_2(t) \), and \( d1T_3(t) \).

\[
\begin{align*}
d1T_1(t) &= \frac{1}{4}m*a^2*\omega^2*cos(\theta)*sin(\theta) \\
d1T_2(t) &= 0 \\
d1T_3(t) &= 0
\end{align*}
\]

The solution indicates that there exists one nonzero component of the net torque in the body reference frame of the disc.

### 3.5 Related Research

In this section we review the work related to automated modeling and/or simulation of physical systems, and compare them to ours.

#### 3.5.1 Model Generation

Falkenhainer and Forbus [1991] describe a form of compositional modeling where a device model is automatically formulated by composing a set of relevant model fragments which are initially obtained by matching the terms of a query to a domain theory and elaborated later. There are several differences between our work and theirs. In ORACLE we distinguish model fragments from entities; model fragments are used for describing physical phenomena and entities for objects (both composite and primitive); model fragments are indexed by the "mf" slot of an entity. In Falkenhainer and Forbus' approach, model fragments are used for describing all the phenomena, objects, and devices, and are organized into mutually exclusive sets called assumption classes. When the class condition holds, one and only one of the assumptions associated with the class must hold and the model fragment containing that assumption must be included in a model. Once the model fragments with appropriate assumptions are selected, the process
of instantiating the model fragments and assembling them is straightforward. While a composite object in Oracle can consist of any heterogeneous parts, a unique minimal covering of parts taken from a single part-of hierarchy is required to exist in Falkenhainer and Forbus' approach to generate a simplest possible model. They do not have a capability of handling detailed structural relations among parts and choosing appropriate reference frames for parts, and therefore cannot handle complex motions such as motion of multiple objects. For behavior generation Falkenhainer and Forbus use either qualitative simulation by QPE [Forbus, 1990] or quantitative simulation by numeric simulation, whereas we use numeric simulation or analytic method.

In [Nayak, 1992a; Nayak, 1992b; Nayak et al., 1992], Nayak describes a method to construct a device model by selecting an appropriate model for each component of the device using structural, behavioral, and expected behavioral constraints. In his system, a model is formulated by composing a set of model fragments, as in ours. However, the uses of the models produced by the two systems are different. While Oracle constructs a model to predict motions of physical systems, his system builds a model to explain causal relations between parameters of a device. Another difference is that he uses order of magnitude reasoning for behavior generation while we use numeric simulation. His order of magnitude reasoning method is restricted to generating the behavior at a fixed point in time, but we can predict the behavior changing with time as well as the behavior at a fixed point in time.

The Sigma system developed at NASA Ames Research Center [Keller and Rimon, 1992; Keller et al., 1994] is a tool which aids a scientist-user in building a model. After the interaction with the user, it produces a model specified in data flow graph and executes the model to compute a unknown quantity. Like Oracle, Sigma organizes and represents domain knowledge in frame structures.
However, it is a user-assistant system rather than an autonomous model-building system, and has several restrictions in constructing and executing a model. which Oracle does not have. For example, multiple quantities cannot be computed at the same time because it cannot solve more than one equations simultaneously, and the types of equations are restricted to algebraic equations or first-order ordinary differential equations; model fragments cannot be put together in an arbitrary order due to the strict backchaining control strategy of its model building process. It converts the input values into a common, consistent set of scientific units, but does not have a provision to transform a vector quantity measured in one reference frame to another, which is necessary in dealing with moving objects.

The MSG system developed by Ling et al. [1993] generates mathematical models for analyzing heat transfer behavior. The approach of the MSG system to building a model is similar to that of Oracle in the sense that it is compositional. However, there are several differences. While Oracle focuses on modeling physical systems involving motion, MSG models physical systems involving heat flow. Therefore, the domain knowledge the two systems use are different. A second difference is that Oracle represents the knowledge explicitly in general, declarative form, but much of the knowledge that the MSG system uses is embedded in the system as part of its algorithm. A third difference is that, while Oracle generates a model and then solves the equations of the model to predict the behavior of a given physical system, MSG presently does not solve the equations of its generated model.

3.5.2 Simulator Generation

Yet another related works concern simulation generation instead of model generation. The SimLab system [Palmer and Cremer, 1991] produces a simulator from a user-provided physics model. Given a mathematical model of a physical
phenomenon and instructions for solving the resulting equations, SimLab transforms the model into an executable simulation code to analyze the phenomenon. However, the user still has the burden of creating the mathematical model. The program built by Berkooz et al. [1992] is similar to SimLab. It is basically a "compiler" for translating differential equations expressed in mathematical and programming constructs into an executable code. The SINAPSE system [Kant, 1991; Kant, 1992] also automatically transforms a given model into a program in desired language, which is either Fortran, Connection Machine Fortran, or C, though again the human user must create the input model.

A number of mechanical device simulators are commercially available, such as Adams [Orlandea et al., 1977; Dawson, 1985], DADS [Haug, 1989], and others (see surveys in [Fallahi and Ragsdell, 1983; Haug, 1984; Paul, 1979; Potter, 1992]). These programs, like most simulators, incorporate physics knowledge such as Newton’s laws of motion phenomena directly into algorithms rather than representing it explicitly. The simulators include powerful algorithms for forming and solving the equations of motions for a wide variety of mechanisms, but lack the flexibility that Oracle has to explicitly instantiate general model fragments in particular situations.

3.5.3 Other AI Research on Physics

Early AI works built programs which could solve textbook physics and mechanics problems. The ISAAC program [Novak, 1977] solves physics problems stated in English. The English sentences of the problem statement are transformed into a number of different representations, many of which are based on frames [Minsky, 1975] with procedures specialized for particular tasks. The use of specialized representations and procedures simplifies many problem solving processes, but requires the translation between the various representations whenever necessary.
Like ISAAC, NEWTON [de Kleer, 1977] uses multiple representations and the knowledge is stored in frames, each containing related physical formula. NEWTON solves a set of roller coaster problems – problems about the motion of a particle on a path. It works in two stages: qualitative reasoning (envisionment) and quantitative reasoning. Given an initial position of a particle and the shape of the path, an envisionment tree of all possible scenarios is generated using a set of production rules. The ambiguity associated with each branch of the tree is later resolved by quantitative tests.

MECHO [Bundy, 1978] extends some features of NEWTON. MECHO applies the same technique and same representation to any problem, and can still solve not only hard problems but also easy problems. It generates only those branches required to answer the questions asked instead of exploring all paths of an envisionment tree. Like NEWTON, MECHO works in two stages, qualitative test followed by quantitative test. The qualitative test consists of four questions about the path slope (up or down), the path concavity (concave or convex), the direction of the particle motion (left or right), and the position of the particle relative to the path (above or below). Once the problem passes the qualitative test, quantitative test involving simple equations and inequalities is done. MECHO is similar to ORACLE in the sense that it uses the same problem-solving method and same representation to any problem and that it does not generate all possible scenarios to solve a problem. However, it is different from ORACLE in many ways. ORACLE does not start with qualitative test to solve a problem. More importantly, the degree of complexity of motions handled by the systems is very different. All the above systems (ISAAC, NEWTON, and MECHO) cannot deal with complex motions in three dimension; both NEWTON and MECHO are designed to solve problems concerning a single object sliding on a curve in two dimension. The problems solved by ISAAC are also simple physics problems in one or two dimensions. This makes the problem solving of the programs simple
because they do not have to deal with vector quantities in three dimension or moving multiple reference frames.

There are more recent qualitative physics approaches which can reason about motion of physical systems. They usually take as input a structural description of a mechanical device, and produce time-varying motion of the device as output. However, their capability of reasoning about motion is limited to simple motions only, and their prediction of motion is ambiguous. For example, many qualitative physics works which can solve the linear harmonic oscillator problem [Forbus, 1984; Kuipers, 1986; Struss, 1988; Weld, 1988; Williams, 1986] cannot handle the more complex motions, such as those of the spring-block systems described in this chapter because qualitative variables are not adequate to handle spatially complex motions.

Previous AI research in spatial reasoning about mechanical devices [Faltings, 1987b; Gelsey, 1989; Gelsey, 1994; Joskowicz and Sacks, 1991] has devoted considerable attention to reasoning about contacts between solid bodies, a problem ORACLE does not presently address. Like the commercial simulators, these programs incorporate knowledge of physical phenomena directly into algorithms rather than attempting to explicitly instantiate general model fragments in particular situations, as ORACLE does.

3.5.4 Model Selection

Another relevant line of work concerns model selection rather than model generation. The Graph of Models (GoM) framework of Addanki et al. [1989; 1991] facilitates the selection of an appropriate model from alternative models, which are generated \textit{a priori} and organized in a graph. In the GoM framework, each node represents a model of a system being analyzed and each edge indicates assumption changes from one model to the other. A model in this graph has
acceptable accuracy if its predictions are free of conflicts. When a conflict is detected either empirically or internally, a set of domain-dependent parameter change rules is used to move to a new model. Search begins with the simplest model in the graph, and terminates when an accurate enough model is found. One of the problems with the GoM approach is that the space requirement of GoM increases exponentially in the number of assumptions because it must explicitly store all possible combinations of assumptions. Another, but related, problem is that a huge data structure of GoM must be created before starting any analysis.

In [Weld, 1992], Weld introduces an approach to model selection which does not require a graph of models to be explicitly predefined. His framework reasons about model accuracy based on model sensitivity analysis, and performs not only model refinement but also model simplification using the model sensitivity analysis. Model sensitivity analysis is performed efficiently when a model is a fitting approximation of the other (P is a fitting approximation of Q if Q contains an independent parameter, called a fitting parameter, such that the predictions by Q approach the predictions by P, as the fitting parameter approaches a limit). When all the approximations are fitting approximations, the domain-dependent parameter change rules can be replaced by a domain-independent method for model switching.

Ellman et al. [1993] introduce a method called gradient magnitude model selection to guide model selection in the sailboat design problem. The method has been used to find the simplest model in hillclimbing search.

3.5.5 Model Approximation

Yip [1993] describes a method to formulate an approximate model from a given detailed model based on the theory of asymptotic order of magnitude. He simplifies a model by examining the limiting cases where the model becomes singular.
The **IDEAL** system [Falkenhainer, 1993] is similar to that of [Yip, 1993] in that it derives a simplified model from a given detailed model. But they differ in that **IDEAL** uses two approximation operators (what he calls dominance-reduction and iso-reduction) instead of order of magnitude reasoning and produces each simplified model's credibility domain as well, which specifies the range of model parameter values for a given error tolerance.

### 3.5.6 Model-Based Application

While we have focused on methods for building and simulating a model, there are several task-specific works which make use of models. For example, [Hunt and Price, 1991; Price and Hunt, 1991] describe a model-based diagnostic system which uses a functional model and a structural model to decide possible failure modes, the outcome of the failure modes, and the final effects of the failure mode. However, the way to integrate the functional model and the structural model models is not general, and specific to the task of diagnosis of the screenwash system of a car. Cagan and Agogino [1987] describe a method called 1stPRINCE which uses a mathematical model to derive new design features of geometric and material properties from previous designs. 1stPRINCE is similar to ORACLE in the sense that it reasons from fundamental physical equations and concepts instead of pre-compiled knowledge of a human designer. But the implementation of 1stPRINCE is limited to the design of a beam under torsion, which must be represented in polar coordinates in one dimension.

### 3.6 Directions for Further Research

There are several directions in which ORACLE can be extended. They are grouped into the following classes:
1. problems which require no change to the current work

2. problems which require minor change to the current work

3. problems which require major change to the current work

**Problems Which Require No Change**

In this chapter we showed examples of how ORACLE predicts the behavior of single or multiple blocks connected by spring(s). When the blocks are replaced by other types of rigid bodies such as cylinders or wedges, ORACLE can still predict their motions. It can also compute the state variables or forces of a moving physical system consisting of rigid bodies. These problems can be solved by applying Newton2, Euler, and/or multiple moving reference frames. For example, when a propeller-driven airplane is reasoned about using an approximate model, forces which the propeller must sustain for the propeller-driven airplane at constant angular velocity can be obtained using Euler’s equations and multiple reference frames (see pages 449 – 450 of [Bradbury, 1968] for the detailed description of this problem).

**Problems Which Require Minor Change**

In the current implementation of ORACLE, there are only certain classes of entities and model fragments available in the knowledge base. Adding more model fragments and entities would expand the types of physical systems covered by ORACLE. It would also be a valuable test for the extensibility of the system.

Problems which do not involve new physical phenomena, but require model fragments with different assumptions or representations will involve minor changes. For example, the two model fragments for spring forces presently assume a linear spring with linear damping. To model nonlinear springs, we need to add a new
model fragment with different equations and assumptions. The types of springs should be considered when an ifadded procedure chooses between the two model fragments of spring forces. The deceleration of an automobile by a drag parachute (a problem described on pages 447 – 448 of [Fox and McDonald, 1992]), can also be reasoned about using the model fragments of drag and Newton2. But the equation of the drag model fragment needs be represented using the coefficient of drag.

\[ F_D = \frac{1}{2} C_D \rho V^2 S \]

where \( F_D \) is drag force, \( C_D \) is drag coefficient, \( \rho \) is fluid density, \( V \) is fluid speed, and \( S \) is the total surface area in contact with fluid.

Some spatial reasoning problems can be solved by qualitative interpretation of the quantitative models produced by Oracle. For example, qualitative description of motions (such as translational, rotational, oscillatory, or tumbling) can be easily obtained by postprocessing the simulation results of the models. Coverage of space of a moving object, any regularity of the coverage over time (such as monotonically decreasing coverage of a damped spring), or possible contact/collision with other moving objects (intersection of the coverages during same time intervals) can also be produced by postprocessing the simulation results. Alternative method of producing qualitative description of motions is possible by analyzing the variable dependency relation among model fragments instead of generating quantitative models and then postprocessing them. This method has advantage of avoiding complex computation to solve simple problems, but requires using different problem-solving techniques and representations to different types of problems.
Problems Which Require Major Change

When something goes wrong during problem solving (e.g., a model cannot be solved due to fewer equations than unknowns), Oracle currently prints the dead-end situation, asks for further information, and quits. The user has to figure out the cause of the problem and rerun the program with new information. In order for Oracle to suggest possible directions to fix the problem, it must have a capability of reasoning about equations and unknowns.

Oracle has been tested to predict the motion of man-made systems such as mechanical systems. However, Oracle can be used to model the motion of a molecule in solution if the three-dimensional structure of the molecule and the mathematical formulation for the forces between the atoms of the RNA are known. The way Oracle reasons about molecules in continuous motion is the same as it does for mechanical systems; the total force on each atom of the molecule is computed, the force is then substituted in the equations of the Newton2 and/or Euler model fragments, the model fragments are composed to construct a model, and the model is numerically simulated to give the state variables of every atom as a function of time. Unfortunately, this kind of molecular dynamics simulation is not possible until we know the three-dimensional structures and the forces of the molecules, which are not known for most RNAs at present. Some molecular dynamics simulations have been carried out to study proteins and segments of DNA by applying Newton’s second law of motion to the atoms of the molecules [Kuntz, 1987; Levin, 1992]. Although a molecular dynamics simulation is simple in concept, it is limited in practice by the computing power because such computation involves tens of thousands atoms. Therefore, efficiency in implementation will also become an important issue in making a molecular dynamics simulation useful.

There are several problems raised in scaling up Oracle not only to broaden
the types of physical systems to be modeled but also to model a physical system with a large number of components and phenomena involved.

First, the size of a model generated by the current version of Oracle can restrict scaling up either because of practical limitations of solving a huge model by Maple or C simulator or because solving a huge model takes too much time to be useful. Table 3.1 shows the sizes of the models and the times for formulating and solving the models for the examples shown in this chapter. The model sizes of spring-block systems with different configurations from the examples in Table 3.1 are about the same as those of the spring-block systems with the same number of blocks and springs in Table 3.1, that is, the model sizes of spring-block systems are independent of their configurations. As we can see in Table 3.1, the size of a model is not directly proportional to the number of model fragments instantiated. Rather it is proportional to the number of unknowns in the equations of a model, or to the number of variables specified in the problem statement (except for the disc-rod example).

As described earlier, a model contains variables, assumptions, names of model fragments, and equations. The equations of a model are composition of the equations of the model fragments instantiated to construct the model. Although the equations of most model fragments exist in a very short, simple form before instantiation, they may become very long and complex after they are instantiated for the particular entities and physical phenomena in the problem. This explains the large variations in the sizes of models with similar number of model fragments; the big size of a model is attributed to the long equations of instantiated model fragments. Even for a same model fragment, the equations after instantiation can be very different in their sizes depending on for which variables they are to be solved and which variables of the equations are known. For example, the equations of the model fragment Euler are as follows:
<table>
<thead>
<tr>
<th>example</th>
<th>#variables</th>
<th>#unknowns</th>
<th>#model fragments</th>
<th>#lines/#chars</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>model_gen</td>
<td>model_sol</td>
<td>display_save</td>
<td>total</td>
</tr>
<tr>
<td>disc-rod</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3/185</td>
</tr>
<tr>
<td></td>
<td>Euler (1)</td>
<td>2.766</td>
<td>0.167</td>
<td>3.083</td>
</tr>
<tr>
<td>algebraic model</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>30/2299</td>
</tr>
<tr>
<td>of boat</td>
<td>FDrag (2), Lift (2), Newton2 (1)</td>
<td>4.900</td>
<td>4.616</td>
<td>4.917</td>
</tr>
<tr>
<td>differential</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>31/2361</td>
</tr>
<tr>
<td>model of boat</td>
<td>Newton2 (1), FDrag (2), Lift (2), Lift_at_zero_speed (1)</td>
<td>4.700</td>
<td>18.700</td>
<td>4.716</td>
</tr>
<tr>
<td>one</td>
<td>12</td>
<td>12</td>
<td>3</td>
<td>206/16260</td>
</tr>
<tr>
<td>block system</td>
<td>Newton2 (1), Euler (1), Springforce2 (1)</td>
<td>14.583</td>
<td>19.800</td>
<td>14.833</td>
</tr>
<tr>
<td>two</td>
<td>24</td>
<td>24</td>
<td>6</td>
<td>2010/159177</td>
</tr>
<tr>
<td>block system</td>
<td>Newton2 (2), Euler (2), Springforce1 (1), Springforce2 (1)</td>
<td>108.533</td>
<td>118.283</td>
<td>110.433</td>
</tr>
<tr>
<td>three</td>
<td>36</td>
<td>36</td>
<td>10</td>
<td>5330/421920</td>
</tr>
<tr>
<td>block system</td>
<td>Newton2 (3), Euler (3), Springforce1 (2), Springforce2 (2)</td>
<td>339.350</td>
<td>264.616</td>
<td>344.767</td>
</tr>
</tbody>
</table>

Table 3.1: Size of a model and time for formulating and solving it for each of the ORACLE examples. The units of the CPU times are in seconds.

#variables the number of variables in component form, specified in the problem statement

#unknowns #variables plus the number of newly generated variables in the equations of a model

#model fragments the total number of model fragments instantiated for a model

#lines/#chars counts of lines and characters of a Maple-formulated model

model_gen time CPU time for generating a model

model_sol time CPU time for solving the equations of a model (for the spring-block systems, the models were solved by C simulators)

display_save time CPU time for displaying the result of solving a model and for saving all the results

total time the total CPU time for generating, solving, and displaying a model
Figure 3.35: Size of a model as a function of #unknowns in the equations of the model. For the count of lines of a model with 3 unknowns, the average value of the line counts of the 3 models with 3 unknowns (disc-rod, algebraic model of boat, differential model of boat) is used.

Figure 3.36: Total CPU time in seconds for generating a model and solving it as a function of #unknowns in the equations of the model. For the time of a model with 3 unknowns, the average value of CPU time of the 3 models with 3 unknowns (disc-rod, algebraic model of boat, differential model of boat) is used.
\[
I_1 \frac{d\Omega_1(t)}{dt} + (I_3 - I_2)\Omega_2(t)\Omega_3(t) = K_1(t)
\]
\[
I_2 \frac{d\Omega_2(t)}{dt} + (I_1 - I_3)\Omega_3(t)\Omega_1(t) = K_2(t)
\]
\[
I_3 \frac{d\Omega_3(t)}{dt} + (I_2 - I_1)\Omega_1(t)\Omega_2(t) = K_3(t)
\]
where \(I_1, I_2,\) and \(I_3\) are the principal moments of inertia of inertial tensor, \(\Omega_1(t),\) \(\Omega_2(t),\) and \(\Omega_3(t)\) are the three components of the angular velocity, and \(K_1(t),\) \(K_2(t),\) and \(K_3(t)\) are the three components of the total torque. Solving the above equations for the variables \(K_1(t), K_2(t),\) and \(K_3(t)\) is straightforward when all the other variables are known as in the disc-rod example; it only requires computing the left hand sides of the equations to assign values to the components of torque. However, solving the equations for the variables \(\Omega_1(t), \Omega_2(t),\) and \(\Omega_3(t)\) when the value of torque is unknown, as in the spring-block examples, is much more complex. We have to derive the function of torque by vector cross product \((\mathbf{K}(t) = \sum \mathbf{r} \times \mathbf{f}(t))\), transform the differential equations into canonical forms, and then solve them by numeric integration. In general, when the model composition process involves vector cross products or multiplications of matrices with vectors, as in the spring-block system examples, ORACLE presently produces a very large Maple model even though the resulting equations are simplified by applying Maple function “simplification”.

Figure 3.35 shows how fast models grow with the number of unknowns of the models. The size of a model is measured in terms of the number of lines of the model. Using the count of characters instead of lines gives a similar growth rate in this case. For the count of lines of a model with 3 unknowns, the average value \(((3 + 30 + 31)/3 = 21.3)\) of the line counts of the 3 models with 3 unknowns (disc-rod, algebraic model of boat, differential model of boat) is used in the plot. When the number of unknowns increases from 3 to 12, the model size increases
Figure 3.37: Comparison of the size growth of a model with the time growth for formulating and executing a model. The units of the model sizes are in the line counts of models and the units of the times are in seconds.

from 21.3 lines to 206 lines, which is almost 10 times. This is because the spring-block example involves much more complex computation which results in long equations. It is also notable that when the number of unknowns increases from 12 to 24, the model size increases from 206 lines to 2010 lines, again about 10 times, and that when the number of unknowns increases from 24 to 36, the model size increases from 2010 lines to 5330 lines. Given limited data, the growth rate of a model is roughly quadratic in the number of unknowns. While Figure 3.35 shows the growth rate of the mode size, Figure 3.36 shows the growth rate of the time for modeling and simulation. The total time for generating a model and solving it is also proportional to the number of unknowns of the model, but the growth rate of time is not as fast as that of the model size. (see Figure 3.37 for the comparison).

Some of the large models have long equations which cannot be simplified further in their nature unless we decide to produce approximate models instead.
However, some models may be simplified without losing accuracy of their predictions by doing additional processing on the equations instead of applying Maple-built-in simplification functions. Reformulating the model generation and solving process of Oracle is another direction to consider in order to efficiently construct and solve a large model. Restricting Oracle to a certain type of physical system is another way to scale up the modeling system to handle a complex physical system with a large number of components and phenomena involved. For example, if we want a special purpose modeling system for spring-block systems only, we can make the modeling system generate a single model that works for any number of blocks. As the number of blocks increases, the model would need more data storage but the model itself does not become larger. This kind of approach to scaling up a modeling system has advantage of being able to model and simulate a complex physical system without the size problem of a generated model, but has disadvantage of losing the breadth of physical systems covered by the modeling system.

Second, selecting relevant model fragments would become a more important issue in scaling up. Including additional properties of an object, such as electrical or thermal properties, in the description of an entity as slots does not cause a problem in selecting model fragments because when Oracle determines the relevance of a model fragment to a given problem it checks whether each variable of the model fragment is mentioned as an entity property or variable of the problem statement or derivable from them. The difficulty is in selecting a model fragment among multiple model fragments with same or similar variables but with different assumptions. In the current implementation of Oracle, model fragments are indexed by the mf slots of entities, with more frequently relevant ones first, and therefore the search process is sensitive to the order of the model fragments. We may want a more efficient method for organizing model fragments which will facilitate identifying and retrieving relevant model fragments.
The third problem in scaling up is related to the size of a generated model. As a model gets bigger, it would become more difficult to understand the model or to validate its solution. Having Oracle provide an explanation of its solution, or checking the solution against that of an approximate model or experimental data will help understand or validate the model.

3.7 Summary

Models of general motions in three dimension are central in many areas but difficult to formulate by hand. This chapter has described a framework for automating the model-building process for physical systems with multiple moving objects in arbitrary configurations, and an implemented system called Oracle. Given a description of a problem involving a moving physical system, Oracle automatically identifies relevant model fragments, instantiates them for the particular entities and physical phenomena in the problem, composes the instantiated fragments to form a model, and simulates the model to solve the problem. Knowledge of physical phenomena is represented with model fragments which can be shared and reused by many models. Most of the knowledge is just the same fundamental equations that appear in any standard mechanics textbook, with their implied semantics of vectors and frames of references. Starting with the most basic, simple concepts in the domain of mechanics, Oracle can still generate a powerful model for complex motions. This is a new method which solves several problems with existing AI modeling work on motion by: (1) declarative, algorithm-neutral representation of physics knowledge; (2) simultaneous handling of multiple equations (algebraic or differential, linear or nonlinear); and (3) explicit handling of vector quantities and frames of reference. Oracle seems to be a useful tool kit for generating models from a library of reusable model fragments for different types of physical systems involving spatially complex motions.
As discussed in Section 3.5, there are many programs developed for reasoning about motion of mechanical devices. However, the programs do not solve problems in a general method from basic physics principles, but rely on specific methods specialized for certain classes of problems. Letcher [1976], for example, uses a numerical procedure adapted from Newton-Raphson iteration to find the optimum sailboat velocity with the maximum component in the wind direction, or the sailboat velocity which will balance all the forces. ORACLE solves the same sailboat problems without requiring special-purpose problem solving methods, as it does for other mechanical devices.
Chapter 4

Modeling Folding

In this chapter, we discuss predicting the folding structures of a biological system, RNA. RNAs are not found in nature as simple chains of bases. In their biologically active form, they are folded upon themselves, forming complex structures. Determining the folding structure of an RNA molecule from its underlying linear structure is a complex problem involving both spatial reasoning and the use of knowledge of biology and chemistry.

The folding structure can be determined directly by X-ray crystallography, but most RNAs cannot be crystallized with current technique. Since the structure of RNA determines its biological activity, predicting the RNA structure is important for understanding the RNA. For an RNA of \( N \) bases, the number of different base pairs is \( N(N - 1)/2 \). Each base pair can belong to a helix or not. Thus, the total number of possible structures is \( O(2^{N^2}) \). An exhaustive search for every possible structure is inefficient and intractable for this problem. We use a heuristic algorithm with folding constraint propagator to construct models of structures. Like the models of motion, the structural models are composed of model fragments which are searched by the algorithm. The searching process of the algorithm is guided by scoring functions whose parameters are obtained from several sources of knowledge. Since none of these sources alone provides complete and accurate knowledge for the task, generalized knowledge from the sources is incorporated into our model. As a result, we use a simple model, which is a combination of rough approximation of the original knowledge, rather than relying on any
single source of knowledge as the sole determinant of the prediction. This simple, approximate model achieves much better performance in the prediction than a complex model.

4.1 Background

4.1.1 RNA

RNA, like DNA, is a long polymer consisting of nucleotides, each composed of a base, a sugar, and a phosphate group. The covalent structure of RNA differs from that of DNA in two respects. As indicated by its name, the sugar units in RNA are riboses rather than deoxyribooses. Ribose contains a 2' - hydroxyl group not present in deoxyribose. The other difference is that one of the four major bases in RNA is uracil instead of thymine. The backbone (invariant part) of RNA consists of sugars and phosphate groups. The variant part of RNA is its sequence of four kinds of bases: adenine (A), cytosine (C), guanine (G), uracil (U). It is the sequence of bases in the chain that distinguishes one type of RNA from another. Therefore, the linear structure of RNA can be concisely represented in a sequence of the bases (A, C, G, U). RNA molecules are usually single-stranded, except in some viruses. Single-stranded RNA often folds back onto itself in structures stabilized by hydrogen bonds between A and U and between G and C. The A-U and G-C base pairs are called canonical pairs (Table 4.1). Minus sign (−) indicates a gap in the sequence in Table 4.1. The so-called G-U wobble pair is also frequently found although it may have a weaker bond than the G-C canonical pair. Between any two bases we consider these two types of pairings and consider all the other types as mismatches.
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>U</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td>canonical</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>canonical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>canonical</td>
<td></td>
<td></td>
<td>wobble</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>canonical</td>
<td>wobble</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Types of base pairings between two bases

4.1.2 Nomenclature of RNA structures

Three levels of structure are frequently cited in discussions of RNA architecture. Primary structure of RNA refers to the base sequence of the RNA. Secondary structure refers to two-dimensional conformation and bonding pattern formed by hydrogen bonds between complementary bases. Tertiary structure is the three-dimensional conformation. It is the tertiary structure of RNA which completely determines its biological activity. Unfortunately, precise determination of tertiary structure is not possible without detailed three-dimensional structural data. Transfer RNA (tRNA) is the only type of RNA which has ever been crystallized [Kim et al., 1974], and three-dimensional structure data are not available for most types of RNAs. However, the secondary structure of RNA can be predicted largely by its base sequence and biochemical or mutational support. The work of this dissertation deals with the secondary structure. The secondary structural model of an RNA molecule not only provides a source of insight into how the RNA functions, but also is essential in determining its tertiary structure.

The secondary structure of RNA is a combination of several substructures each matching one of elementary structure motifs. Figure 4.1 shows a hypothetical RNA molecule folded in a structure containing each of the structure motifs that are known to occur in actual RNA molecules. The secondary structure consists of single-stranded parts and double-stranded parts. In double-stranded parts, also
Figure 4.1: Terminology of elementary structure motifs of secondary structure called helices, stem or duplexes, the molecule folds into a double helix in which ribonucleotides pair and adjacent pairs stack one upon the other. Helices, labeled h1-h5 in Figure 4.1, can be as short as 2 base pairs.

Single-stranded parts can be divided into five different types. A hairpin loop is a loop of unpaired ribonucleotides bridging a helix (labeled HL in Figure 4.1). Hairpin loops are known to bind proteins, form tertiary interactions, and serve nucleation sites for RNA folding. The smallest loop capable of bridging a helix was originally believed to have a minimal length of three bases because a hairpin loop with less than three bases makes a too sharp turn to be stable [Tinoco et al., 1973]. But there are some recent studies reporting a hairpin loop of two unpaired ribonucleotides [Cheong et al., 1990]. A bulge loop or bulge is defined as unpaired ribonucleotides on one strand of a double-stranded region where the other strand has continuous base pairing (BL in Figure 4.1). Bulge loops range in size from one to many ribonucleotides. An internal loop contains at least one unpaired ribonucleotide on each strand (IL in Figure 4.1). A multiple loop, junction or multi-branched loop contains three or more helices with a variable number of
unpaired ribonucleotides where the helices come together (ML in Figure 4.1). A dangling end consists of unpaired ribonucleotides at the 5' or 3' end (D in Figure 4.1).

In the RNA folding structure, each structure motif previously defined contributes to the total structural stability. Helices formed by consecutive base pairings contribute to increased stability, while single stranded parts destabilize neighboring helices. Many experiments have been carried out with chemically synthesized RNAs to obtain the structural stability measured by the loss in free energy in kcal/mol [Tinoco et al., 1973]. A data set of free energy values describing the contribution of each structure element is called an energy model. Several energy models are available but none of them can be considered perfect. Predicted structures are often radically different if a different energy model is used or a slight change is made to the energy values.

### 4.1.3 Existing Theoretical Approaches to RNA Structure Prediction

There are two major theoretical approaches to predict RNA secondary structures. One is energy minimization method and the other is phylogenetic comparative method. The energy minimization method uses a dynamic programming algorithm combined with an energy model to determine folding structures with minimum or near minimum free energy [Sankoff et al., 1983; Zuker and Stiegler, 1981; Zuker, 1989b]. Let $M_{i,j}$ be the maximum number of base pairs in the sequence segment from base $i$ to base $j$. Take $M_{i,j}$ known for $0 \leq i, j \leq n$. Then add base $n + 1$. If base $n + 1$ is unpaired in the optimal structure, $M_{i,n+1} = M_{i,n}$. Otherwise, base $n + 1$ is paired to base $j$, where $1 \leq j \leq n - 2$ because there must be at least 2 bases in an end loop. Then

$$M_{1,n+1} = M_{1,j-1} + 1 + M_{j+1,n} \quad (4.1)$$
The 1 in the above equation counts the new base pair between \( j \) and \( n + 1 \), while
the other two terms are optimal for the other two segments of sequence. To collect
this into a recursion, let \( \delta_{i,j} = 1 \) if bases \( i \) and \( j \) can pair and 0 otherwise. Then

\[
M_{1,n+1} = \max(M_{1,j-1}, \max_{1 \leq j \leq n-2} (M_{1,j-1} + 1 + M_{j+1,n})\delta_{i,j}) \quad (4.2)
\]

The complication of implementing the algorithm comes from handling free
energy values associated with various types of structure elements. Limitations of
the energy minimization method are partly due to the uncertainty of the underlying
energy model. As we show later, the method is also overly sensitive both to
point mutation and to range of a sequence.

On the other hand, the phylogenetic comparative method is different from
dynamic programming. There is no minimum energy calculations made. The
method desires to find many common helices of similar sequences, which will
determine secondary structures. The underlying assumption of the phylogenetic
comparative method to predict RNA structures is that preservation of a structure
between homologous sequences, despite changes in their base sequences, is a sup-
port for the structure’s existence. Homologous sequences have common ancestry
and function, hence they are expected to have similar higher-order structures. For
the phylogenetic comparative method, the sequences of homologous RNAs from
different organisms must be aligned so that evolutionary homologous residues are
juxtaposed, commonly in columns (see the example below). The homologous se-
quences compared must be sufficiently different to provide numerous instances of
sequence variation with which to test pairing possibilities. But, if they differ too
much, they cannot be aligned with confidence.

Consider, for example, an alignment of four imaginary RNA sequences below,
where sequence 1 contains a subsequence GAGCU. GAGCU has the potential to
base pair with any AGCUC within sequence 1 and such a subsequence may occur
several times in that sequence. In order to identify which occurrence of AGCUC
actually pairs with GAGCU, the homologous RNA sequences from different organisms are examined.

sequence 1: GAGCU ... AGCUC ... AGCUC ... AGCUC
sequence 2: GAUCU ... AGCUC ... AGCUC ... AGAUC
sequence 3: GAUCU ... AGCUC ... AGCAC ... AGAUC
sequence 4: GACC ... AGCUC ... AGCAC ... AGGUC

In the above example, only the last occurrence of AGCUC changes to maintain the ability to pair with GAGCU. Such "compensating base changes" at two positions are considered as evidence of the presence of a helix. The current model of secondary structures of 16S and 23S of rRNA were obtained by Noller, Woese, and their collaborators with this method in conjunction with biochemical experiments [Noller and Woese, 1981; Noller, 1984]. Some of the Noller-Woese procedure lends itself to an explicit computer algorithm [Waterman et al., 1984; Waterman, 1988], but the algorithm depends on storing all the patterns of interest and has limitations to be applied to sequences with potentially large number of patterns. The algorithm by Sankoff [1985] for simultaneously aligning and folding M sequences of length N requires $O(N^{3M})$ time and $O(N^{2M})$ space, and is therefore not practical either in terms of time or space. Recently a Macintosh program for use in phylogenetic comparative analysis has been made available by Brown [1991], which shows the result of the analysis in a covariation matrix. However, secondary structure cannot be determined directly from the covariation matrix, especially when many competing helices are present in the matrix. Manual scrutiny of the matrix for possible secondary structures is a laborious task and can overlook possibilities.

In next sections we describe an algorithm and a computer program for predicting common foldings conserved in all or most homologous RNAs. The program is not as sensitive to point mutation of a sequence as the energy minimization
method and does not require manual examination for covariation or secondary structures as usual phylogenetic comparative analysis does. Since the energy model is believed to be inaccurate as well as incomplete, we do not make commitment to the numeric values of the energy model. Instead, we use approximation of the model and seek other possible sources of knowledge, such as homologous RNAs, biochemical experimental data, and biological property of RNA.

4.2 Problem Formulation

The kinds of problems solved by our system are defined as follows.

**Given:** an alignment of homologous RNA sequences

**Find:** the folding structures conserved in the sequences and the pathways from an initial structure to the final structures as the RNA sequences grow over time. Each structure should satisfy the following folding constraints [Sankoff et al., 1983]:

- If the structure contains base pairings \((i, j)\), neither \((i, k)\) \(k \neq j\) nor \((k, j)\) \(k \neq i\) can be contained in the structure.

- If \(i < j < k < l\), the structure cannot contain \((i, k)\) and \((j, l)\) at the same time.

- Base pairing \((i, i)\) is not allowed for all \(i\).

- The number of unpaired bases in a hairpin loop cannot be less than three.

We make the following assumptions to solve the problem:

1. At least two homologous sequences are available and they are properly aligned.
Alignment:

multiple lines of RNA sequences juxtaposed in columns, where each line consists of
1. RNA sequence name of maximum 5 characters
2. space
3. bases or gaps of the RNA sequence

Figure 4.2: The alignment data structure

2. Homologous RNA sequences have similar folding structures.

The folding structure is described in a discrete model because continuously varying parameters such as bond lengths or angles are not considered; either a hydrogen bond exists between two bases or it does not. Once the connectivity relation is determined, the shape of a secondary structure is unambiguously determined as we show later.

4.3 Algorithm

We have implemented a modeling system of folding, called FOLDER, in the C programming language. The top-level algorithm of FOLDER, sketched in Algorithm 4.1, consists of three parts: (1) problem analysis, (2) model creation, and (3) model execution. We discuss each part in detail.

FOLDER takes as input an alignment of homologous sequences, which has the form of Figure 4.2. As output, it produces secondary structures, represented in a form of a sequence and matching parentheses (see Figure 4.3 for the internal data structure of the secondary structure), and a series of substructures at intermediate stages.

In the first phase, the alignment of homologous sequences is analyzed to identify base pairing relations, and a covariation matrix is created from the result
Secondary Structure:

1. total number of diagonals contained in the structure
2. array of pointers to diagonals contained in the structure
3. base sequence to be used in representing the structure (any sequence in the alignment or the consensus sequence)
4. array of ‘–’, ‘(’, and ‘)’ for base pairing relations of the structure

Figure 4.3: The secondary structure data structure

of the analysis. Given an alignment of \( M \) homologous sequences of maximum length \( N \), Folder examines the sequences and represents the result in an \( N \times N \) covariation matrix.\(^1\) Each entry \((i, j)\) of the covariation matrix represents the relation \( BP(i, j) \), which corresponds to the type of base pairing between base \( i \) and base \( j \) for all the \( M \) sequences in the alignment, not for a single sequence. Since different sequences can have different types of base pairings, the type of canonical or wobble pair cannot be used for this relation. Instead, we define the relation \( BP(i, j) \) as follows:

1. If base \( i \) pairs with base \( j \) in all the sequences and there is no variation in both \( i \) and \( j \), \( BP(i, j) \) is an exact-invariant match (denoted by \( o \)).

2. If base \( i \) pairs with base \( j \) in all the sequences and there is a compensating base change in \( i \) and \( j \), \( BP(i, j) \) is an exact-variant match (\( * \)).

3. If base \( i \) forms a G-U wobble pair with base \( j \) in most sequences, \( BP(i, j) \) is a wobble match (\( w \)).

4. If base \( i \) pairs with base \( j \) in most sequences but not in all the sequences and the frequency of the mismatches does not exceed a specified number,

\(^1\)The covariation matrix produced by Folder has the same format as that of [Brown, 1991], except that Folder’s matrix contains an additional \( BP(i, j) \) relation, a wobble match (\( w \)), that the definition of an inexact match is different, and that a consensus sequence is shown on top of Folder’s matrix.
Algorithm 4.1 Folder's top-level algorithm

Problem Analysis Analyze an alignment of homologous sequences, and represent the result of the analysis in a covariation matrix.

1. Fill in a lower triangular matrix with ‘-’.
2. Determine the relation $BP(i, j)$ for each base pair $(i, j)$ using Algorithm 4.2.
3. Generate a consensus sequence of the homologous sequences using Algorithm 4.3.

Model Creation Search for potential helices from the covariation matrix and compose structural models with them.

1. $\text{best\_score} = -\infty$
2. For each potential helix $h$ identified
   Compute scores $S_1$ and $S_2$ using Algorithm 4.4.
   If ($S_1 > \text{best\_score}$)
     $\text{best\_helix} = h$
     $\text{best\_score} = S_1$ of $h$
3. Find any supporting helix for the $\text{best\_helix}$ using Algorithm 4.5.
4. $\text{skeleton} = \{\text{best\_helix}\} \cup \{\text{any supporting helix}\}$
5. model $M = \text{skeleton}$
6. Detect the helices which conflict with the skeleton using Algorithm 4.6 and remove them.
7. Sort the remaining helices with respect to $S_2$.
8. For each helix $h$ of the sorted helices
   If $h$ fits a current structural model $M$ (i.e., $h$ overlaps or forms a knot with no helix in $M$), add it to $M$ using Algorithm 4.7
   $M = M \cup \{h\}$
   Else, create an alternative structural model $M'$ using Algorithm 4.8
   $M' = \text{skeleton} \cup \{h\}$

Model Execution Simulate the structure formation process by producing structures from a series of submatrix of the covariation matrix at intermediate stages during the RNA sequence growth over time.
$BP(i, j)$ is an inexact match (+).

5. If the frequency of mismatches between base $i$ and base $j$ exceeds a specified number, $BP(i, j)$ is a mismatch (·).

Since the relation $BP(i, j)$ is symmetric, only upper triangular part of the matrix is used. The algorithm for analyzing the alignment and for creating a covariation matrix is outlined in Algorithm 4.2. A consensus sequence of the homologous sequences is also generated using Algorithm 4.3 during the first phase. Upper case letter of the consensus sequence represents a base conserved in all the sequences and lower case represents a base which is not conserved in all the sequences but most representative one in the position.

In the second phase, the covariation matrix is searched for potential helices and folding structures are constructed. A potential helix of RNA appears as a diagonal of non-mismatch symbols (v, *, w, or +) from upper right to lower left in the covariation matrix. Figure 4.4 shows the data structure of the diagonal.

$S_1$ and $S_2$ of Algorithm 4.1 are score functions to find the most stable hairpin loop-stem structure of the overall structure, and to measure the "goodness" of a helix in general positions, respectively. They are weighted sums of five parameters: helix length, number of exact-variant matches, number of wobble matches, number of inexact matches, and the length of a possible hairpin loop created by the helix. Algorithm 4.4 is used to compute the two scores of each helix. A detailed description of the parameter values and the perturbation study of them appear in [Han and Gelsey, 1993].

Once a best_helix with highest $S_1$ score is found, a helix which supports it (i.e., the two helices are directly connected either by an internal loop or a bulge loop) is searched using Algorithm 4.5. The best_helix and its supporting helix (if any) form a skeleton of the overall structure. Helices which violate the folding constraints discussed in Section 4.2 with respect to the skeleton are eliminated.
Algorithm 4.2 Construct a covariation matrix of the relation $BP(i, j)$

Let $A$ be an alignment of $M$ sequences of maximum length $N$
$C$ be an $N \times N$ covariation matrix

For $i = 1, 2, \ldots, N$ do
  For $j = i + 1, i + 2, \ldots, N$ do
    count_match = count_mismatch = count_change = count_wobble = 0
    For $s = 1, 2, \ldots, M$ do
      If $A[s][i]$ forms a canonical pair with $A[s][j]$
        count_match = count_match + 1
      If $s > 1$ AND $A[s][i] \neq A[s-1][i]$ AND $A[s][j] \neq A[s-1][j]$
        count_change = count_change + 1
      Else if $A[s][i]$ forms a wobble pair with $A[s][j]$
        count_wobble = count_wobble + 1
    Else
      count_mismatch = count_mismatch + 1
    If count_mismatch > mismatch_allowed
      $C[i][j] = \text{mismatch } (\cdot)$
    Else if count_mismatch > 0 AND count_mismatch \leq mismatch_allowed
      $C[i][j] = \text{inexact_match } (+)$
    Else if count_match = $M$ AND count_change = 0
      $C[i][j] = \text{exact_invariant_match } (\circ)$
    Else if count_change > count_wobble AND count_mismatch = 0
      $C[i][j] = \text{exact_variant_match } (*)$
    Else
      $C[i][j] = \text{wobble_match } (w)$
Algorithm 4.3 Generate a consensus sequence of the homologous sequences

Let $A$ be an alignment of $M$ sequences of maximum length $N$

For $i = 1, 2, \ldots, N$ do

\[
\text{count}_a = \text{count}_c = \text{count}_g = \text{count}_u = \text{count}_{\text{gap}} = 0
\]

$s = 1, 2, \ldots, M$ do

If $A[s][i] = 'a'$

\[
\text{count}_a = \text{count}_a + 1
\]

Else if $A[s][i] = 'c'$

\[
\text{count}_c = \text{count}_c + 1
\]

Else if $A[s][i] = 'g'$

\[
\text{count}_g = \text{count}_g + 1
\]

Else if $A[s][i] = 'u'$

\[
\text{count}_u = \text{count}_u + 1
\]

Else if $A[s][i] = '-'$

\[
\text{count}_{\text{gap}} = \text{count}_{\text{gap}} + 1
\]

If \[
\text{count}_a = M
\]

\[
\text{consensus}[i] = 'A'
\]

Else if \[
\text{count}_c = M
\]

\[
\text{consensus}[i] = 'C'
\]

Else if \[
\text{count}_g = M
\]

\[
\text{consensus}[i] = 'G'
\]

Else if \[
\text{count}_u = M
\]

\[
\text{consensus}[i] = 'U'
\]

Else if \[
\text{count}_a = \text{max}\ (\text{count}_a, \text{count}_c, \text{count}_g, \text{count}_u, \text{count}_{\text{gap}})
\]

\[
\text{consensus}[i] = 'a'
\]

Else if \[
\text{count}_c = \text{max}\ (\text{count}_a, \text{count}_c, \text{count}_g, \text{count}_u, \text{count}_{\text{gap}})
\]

\[
\text{consensus}[i] = 'c'
\]

Else if \[
\text{count}_g = \text{max}\ (\text{count}_a, \text{count}_c, \text{count}_g, \text{count}_u, \text{count}_{\text{gap}})
\]

\[
\text{consensus}[i] = 'g'
\]

Else if \[
\text{count}_u = \text{max}\ (\text{count}_a, \text{count}_c, \text{count}_g, \text{count}_u, \text{count}_{\text{gap}})
\]

\[
\text{consensus}[i] = 'u'
\]

Else

\[
\text{consensus}[i] = '-'
\]
Diagonal:

1. length of the diagonal in terms of number of bases
2. starting base position of the diagonal in the alignment of the sequences
3. ending base position of the diagonal in the alignment of the sequences
4. $S_1$ score of the diagonal
5. $S_2$ score of the diagonal
6. array of pointers to structures which contain the diagonal
7. alternative diagonal
8. skeleton (TRUE or FALSE)

Figure 4.4: The diagonal data structure

Algorithm 4.4 Compute scores $S_1$ and $S_2$ of a helix

Let $L$ be the length of the helix (number of base pairs)
$E$ be the number of exact variant matches
$W$ be the number of wobble matches
$I$ be the number of inexact matches
$H$ be the length of a hairpin loop formed by the helix

If the user wants to specify the weight values of the 5 parameters for the score functions $S_1$ and $S_2$
Use the weight values specified by the user.

Else
Use default weight values:

\[ w_{11} = w_{21} = 1.0 \]
\[ w_{13} = w_{23} = -0.5 \]
\[ w_{14} = w_{24} = -2.0 \]
\[ w_{12} = 1.0 \]
\[ w_{15} = -0.05 \]
\[ w_{22} = 0.5 \]
\[ w_{25} = -0.01 \]

Compute $S_1$ and $S_2$ by

\[ S_1 = w_{11} \cdot L + w_{12} \cdot E + w_{13} \cdot W + w_{14} \cdot I + w_{15} \cdot H \]
\[ S_2 = w_{21} \cdot L + w_{22} \cdot E + w_{23} \cdot W + w_{24} \cdot I + w_{25} \cdot H \]
Algorithm 4.5 Find a helix which forms a same stem with the best_helix

distance = \infty
For each helix h
    If h is connected with the best_helix by an internal loop
        dist (h, best_helix) = maximum number of unpaired bases on either
        side of the internal loop
    Else if h is connected with the best_helix by a bulge loop
        dist (h, best_helix) = maximum number of unpaired bases of the
        bulge loop
    If dist (h, best_helix) < distance
        distance = dist (h, best_helix)
        support_helix = helix h

Algorithm 4.6 Detect helices which conflict with the skeleton

If helix h forms a knot with the best_helix
OR helix h forms a knot with the support_helix
OR helix h overlaps with the best_helix
OR helix h overlaps with the support_helix
    helix h violates a folding constraint with respect to the skeleton.

using Algorithm 4.6.

Each of the remaining helices is examined, starting with the one with highest $S_2$ score. If the helix can coexist with the helices of the current structures, it is attached to them using Algorithm 4.7. Otherwise, an alternative structure is created for it using Algorithm 4.8. The alternative structure initially contains the skeleton and the helix which had conflicts.

In the final phase, a series of submatrices of the covariation matrix is produced at intermediate stages during the RNA sequence growth. Any $L \times L$ submatrix ($L \leq N$) of the $N \times N$ covariation matrix contains potential helices at a point when the RNA sequence grows to have $L$ bases. Therefore, we take “snapshots” of the structures at a series of intermediate stages for the simulation of the structure
Algorithm 4.7 Add a helix to an existing structure

1. Let $S$ be an existing structure
   $h$ be the helix to be added to $S$
2. for $i = 0, 1, \ldots, h.\text{length} - 1$ do
   $j = h.\text{start} + i$
   $k = h.\text{end} - i$
   $S.\text{pairing}[j] = '($
   $S.\text{pairing}[k] = ')'$
3. $S.\text{no.helix} = S.\text{no.helix} + 1$
4. Include $h$ into the array of pointers to diagonals of $S$
5. $h.\text{no.str} = h.\text{no.str} + 1$
6. Include $S$ into the array of pointers to structures of $h$.

Algorithm 4.8 Create a new structure with helix $h$

1. Let $S$ be a new structure to create
   $h$ be the helix to be included in $S$
   $l$ be the sequence length of the alignment
2. Initialize $S$ by
   For $i = 1, 2, \ldots, l$ do
   $S.\text{pairing}[i] = '-'$
3. For each helix $x$ of the best.helix and support.helix of the skeleton and helix $h$ do
   For $i = 1, 2, \ldots, x.\text{length} - 1$ do
   $j = x.\text{start} + i$
   $k = x.\text{end} - i$
   $S.\text{pairing}[j] = '($
   $S.\text{pairing}[k] = ')'$
4. $S.\text{no.helix} = \text{skelton.no.helix} + 1$
5. Include best.helix, support.helix (if any), and $h$ into the array of pointers to diagonals of $S$
6. best.helix.no.str = best.helix.no.str + 1
7. support.helix.no.str = support.helix.no.str + 1
8. $h.\text{no.str} = 1$
formation process.

The total number of structures produced by the program is restricted to 5 or less. The value of 5 is a somewhat arbitrary choice. Generating more structures by a current version of the program is not difficult at all, but excessive number of possible structures seem to be as uninformative as none. The reason for producing multiple structures instead of one is as follows. First, the program is based on incomplete experimental data, which cause significant uncertainties, and therefore, being able to find several alternative structures is important. Second, RNA molecules may not form a single secondary structure, but may instead have several structures in equilibrium. This has been suggested, for example, for the cIII gene mRNA of bacteriophage λ [Altuvia et al., 1989].

**Folder** finds 1–5 common foldings of M sequences of length N in \(O(MN^2 + N^3)\) time and \(O(N^2)\) space. In the first stage, the covariation matrix is generated in \(O(MN^2)\) time and \(O(N^2)\) space. The number of potential helices in the matrix is \(O(N^2)\) and sorting them with respect to the value of \(S_2\) score (which is a single number) takes \(O(N^2 \log N)\) time. Each of \(O(N^2)\) helices is examined once and compared with helices in \(s\) structures (\(s \leq 5\)). Since each structure contains nonconflicting helices, the maximum number of helices in each structure is \(O(N)\). Constructing common foldings in the second stage takes \(O(N^2 \log N + N^2 \cdot s \cdot N) = O(N^3)\) time and \(O(N^2)\) space, where \(N\) is the maximum length of a sequence (i.e., the maximum number of bases of a sequence). Therefore, the total time and space for the algorithm are \(O(MN^2 + N^3)\) and \(O(N^2)\), respectively. The space requirement of the program is the same as that of the ordinary folding program for a single sequence [Zuker and Stiegler, 1981]. The time requirement is also the same as that of the ordinary program for a single sequence if \(M = O(N)\) or \(M = o(N)\), which is usually the case. For the process modeling, structures at each intermediate stage of \(L\) bases (\(L \leq N\)) can be found in \(O(N^3)\) time and \(O(N^2)\)
space. If the series of structures are obtained by examining the covariation matrix at \( k \) intermediate stages, the total time and space is \( O(k \cdot N^2) \) and \( O(k \cdot N^2) \), respectively. If \( k \) is limited to a constant, process modeling does not raise the overall complexity.

The algorithm has been implemented in C, and graphical representation of RNA structures is shown using LoopViewer. We have successfully tested the program on different types of RNAs (tRNAs, 5S rRNAs, 16S rRNAs, TAR, RRE, and cis-acting packaging sequences of HIV-1, a causative agent of AIDS), in many of which a widely used conventional program fails. A detailed comparison of the results is described in [Han and Kim, 1993].

### 4.4 Examples

#### 4.4.1 tRNA

Consider an alignment of 10 transfer RNA (tRNA) sequences shown in Figure 4.5. The sequences were retrieved from GenBank, and Table 4.2 contains the loci, definitions, and accession numbers of the sequences used for this experiment. Minus signs (\(-\)) in the alignment indicate gaps. The last sequence in the alignment is a consensus sequence of the ten homologous sequences, derived by FOLDER.

FOLDER performs a comparative analysis of the 10 tRNA sequences, and produces a covariation matrix (Figure 4.6). The model creation part of the program searches for all the potential helices (i.e., model fragments). There are only 7 potential helices which are not self-conflicting (i.e., the helix does not contain any base pairing \((i, i)\) and the length of a hairpin loop created by it is at least 3) and which have at least 3 base pairings. Four of them \((1 \rightarrow 7/83 \rightarrow 89, 10 \rightarrow 12/24 \rightarrow 26, 28 \rightarrow 32/40 \rightarrow 44, \) and \(66 \rightarrow 70/78 \rightarrow 82\)) correspond to the well-known 4 helices (acceptor, D, anticodon, T\(\psi\)C) of tRNA [Kim et al., 1974;
Figure 4.5: Alignment of 10 tRNA sequences
<table>
<thead>
<tr>
<th>LOCUS</th>
<th>DEFINITION</th>
<th>ACCESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANITRLCAA</td>
<td>Anacystis nidulans Leu-tRNA-CAA.</td>
<td>K00230</td>
</tr>
<tr>
<td>ANITRLCAG</td>
<td>Anacystis nidulans Leu-tRNA-CAG.</td>
<td>K00231</td>
</tr>
<tr>
<td>BACTRLC</td>
<td>B. stearothermophilus Leu-tRNA-CAA.</td>
<td>M26875</td>
</tr>
<tr>
<td>ECOTRA1A</td>
<td>E. coli Ala-tRNA-1a.</td>
<td>K00139</td>
</tr>
<tr>
<td>ECOTRA1B</td>
<td>E. coli Ala-tRNA-1b.</td>
<td>K00140</td>
</tr>
<tr>
<td>ECOTRL1</td>
<td>E. coli Leu-tRNA-1.</td>
<td>K01550</td>
</tr>
<tr>
<td>ECOTRL2</td>
<td>E. coli Leu-tRNA-2.</td>
<td>K01551</td>
</tr>
<tr>
<td>ECOTRL5</td>
<td>E. coli Leu-tRNA-5.</td>
<td>K00225</td>
</tr>
<tr>
<td>ECOTRS1</td>
<td>E. coli Ser-tRNA-1.</td>
<td>K01555</td>
</tr>
<tr>
<td>ECOTRS3</td>
<td>E. coli Ser-tRNA-3.</td>
<td>K01556</td>
</tr>
</tbody>
</table>

Table 4.2: The Loci, definitions, and accession numbers of tRNA sequences used for the experiment
Figure 4.6: Covariation matrix obtained as a result of comparative analysis on the 10 tRNA sequences
Figure 4.8: tRNA structure drawn by LoopViewer given the input of a structure in a form of matching parentheses
Figure 4.9: Model of the structure formation process of tRNA
Sprinzl et al., 1985], and FOLDER predicts only one structure, represented in a form of a sequence and matching parentheses (Figure 4.7). This is a “quick and easy” representation requiring no special graphics device, and is directly readable by a drawing program such as LoopViewer or LoopDloop\(^2\) for obtaining a better quality representation (Figure 4.8). The first tRNA sequence of the alignment (ANITRLCAA) is used for drawing the structure, and the bases of the tRNA appear in clockwise order.\(^3\)

More than 300 tRNA sequences are known and they have the same cloverleaf folding structure as shown in Figure 4.8 [Sprinzl et al., 1985]. They also share the following features [Stryer, 1988]:

- They are single chains containing between 73 and 93 ribonucleotides each.

- The base sequence at the 3’ end of tRNAs is CCA.

- The base at the 5’ end is usually G.

- About half of the ribonucleotides in tRNAs are base paired to form helices.
  Five groups of bases are not base paired: the 3’ CCA terminal region; the TψC loop, which acquired its name from the sequence ribothymine-pseudouracil-cytosine; the “extra arm”, which contains a variable number of bases; the D loop; and the anticodon loop.

- The anticodon loop consists of seven bases, with the following sequence:

\[
5' - \text{Py} - \text{Py} - \text{X} - \text{X} - \text{X} - \text{modified Pu} - \text{X} - 3'
\]

\(^2\)Macintosh program for visualizing RNA secondary structure. Published electronically on the Internet, available via anonymous ftp from ftp.bio.indiana.edu.

\(^3\)Unlike most other RNAs, tRNA is usually drawn in counterclockwise order in many literatures. The graphical representation of all the RNA secondary structures presented in this dissertation was done by LoopViewer, which always displays secondary structures in clockwise order.
where Py stands for pyrimidines (C or U), Pu for purines (A or G), and X for any base.

- They contain many unusual bases, many of which are methylated or dimethylated derivatives of A, C, G, and U that are formed by enzymatic modification of a precursor tRNA.

For the simulation of the structure formation process, Figure 4.9 shows a series of tRNA structures at ten intermediate stages. In general there are four possible transitions from a structure to another: (1) forming a new helix is initiated (e.g., from stage 2 to 3, from stage 8 to 9, from stage 9 to 10 in Figure 4.9); (2) a helix in a structure is broken down and a new helix is formed (Figure 4.9 does not have this example, but is common in other test cases, for example, from stage 1 to 2, from stage 6 to 7, from stage 7 to 8, and from stage 8 to 9 of the folding process of TAR shown in Figure 4.12); (3) an existing helix is elongated (e.g., from stage 3 to 4); and (4) additional bases are attached to single stranded part (e.g., from stage 1 to 2, from stage 5 to 6, from stage 6 to 7, from stage 7 to 8).

4.4.2 TAR of HIV-1

In the previous section, we showed how FOLDER analyzes homologous sequences of tRNA, constructs a structural model, and simulates the structure formation process. It turned out that tRNA molecules are easy test cases for FOLDER because they have much fewer number of potential helices than other test cases we experimented with. Consider an alignment of 10 TAR (trans-activation-responsive) sequences of HIV-1 (HIVNL43, HIVLAI, HIVHXB2R, HIVMN, HIVJRCSF, HIVSF2, HIVNY5CG, HIVCDC4, HIVHAN, HIVRF) shown in Figure 4.10a. TAR sequences are cis-acting sequences involved in gene regulation, and lies downstream of the transcription start site in the viral long terminal repeat and in the env region
[Cullen, 1990]. Figure 4.10b shows a covariation matrix produced by the comparative analysis of FOLDER. In the covariation matrix there are 63 diagonals which have at least 3 base pairings. All the diagonals are potential helices, but they cannot coexist at the same time due to the folding constraints in Section 4.2. For example, consider a helix (1–5/14–18) and a helix (1–5/19–23) in the covariation matrix of Figure 4.10b. The two helices conflict each other because bases 1–5 cannot pair with both bases 14–18 and bases 19–23 at the same time (the first folding constraint in Section 4.2).

The model creation part of the algorithm first searches for all the potential helices which are not self-conflicting. It then systematically constructs structural models with them. Figure 4.11 shows three possible structures of TAR inferred from the covariation matrix, which are most stable based on the scoring functions with the default parameter values of Algorithm 4.4. The consensus sequence of the ten HIV-1 TAR sequences is used for drawing the structure in the figure. The first structure in Figure 4.11 is identical to the one predicted by a biochemical experiment [Cullen, 1990], which has been corroborated by many other experiments later. Actually the superiority of the first structure to the other two can be determined by looking at them by “eye”. In the second structure, the difference of unpaired bases on either side of an internal loop is too large (11 – 1 = 10), and this will greatly destabilize the adjacent helices. In the third structure, the two helices (GgU/ACC and UCUCUGGUUAG/CUAaCUAGGGA) are identical to those of the first structure, but the other two helices in the third structure are less stable than those of the first structure in terms of the length of a helix and the number of wobble pairs.

For the simulation of the structure formation process, Figure 4.12 shows a series of TAR structures at nine intermediate stages. At stage 7 and stage 9, there are 2 and 3 structures possible, respectively. Transitions from stage 7 to 8
Figure 4.10: (a) Alignment of 10 TAR RNAs of HIV-1 (b) Covariation matrix
and from 8 to 9 seem to be most difficult in terms of the amount of base pairings to be broken down. FOLDER currently finds the most stable structures in each stage instead of the ones which have the easiest transition from the previous stage.

To obtain a stable common structure, it is generally better to look at more sequences. However, if there is little variation between the sequences, a large number of sequences are not necessary. The value of minimum length of a helix greatly affects the number of potential helices. The smaller the value, the more the potential helices. It also influences the prediction. If the number of minimum length of a helix is set to a larger value than 3, then a rough structure is obtained where helices shorter than the threshold value are missing but all the other helices stay. Therefore, increasing the minimum length of a helix does not affect much the prediction other than that shorter helices are ignored. Decreasing the number of minimum length of a helix from 3 to 2 has a slightly more complicated influence on the prediction; it can simply add new helices of length 2 to the previous structure while preserving all the existing helices in it; helices of length 2 can
Figure 4.12: Model of the folding process of TAR
sometimes, however, defeat longer helices in the previous structure and replace them. The value of number of mismatches allowed is another parameter which controls the number of potential helices. The larger the value, the more the potential helices. In most cases we tested, allowing mismatches within 10% (e.g., 1 mismatch for 10 sequences) produces satisfactory results. However, the optimum value of the mismatches allowed will depend on how much variation exists between the sequences and how representative they are.

Weight values of the scoring function do not affect the number of potential helices at all, but have a direct and most influential effect on the prediction. Longer helices are more stable than shorter ones and exact-variant pairs are stronger evidence to base pairing than other types of pairs. Hence, the length and frequency of exact-variant pairs are assigned positive weights in the scoring function. Wobble pairs and inexact pairs are less stable than canonical pairs, and longer hairpin loops are less stable than shorter ones. Therefore, they have negative weights. While decision on the signs of the five parameters is straightforward, decision on the absolute values of them is not. Currently we do not have a general theoretic relation which explains how the relative sizes of the five values are correlated with respect to the prediction. The default parameter values of Algorithm 4.4 have been obtained experimentally.

4.4.3 *cis*-acting packaging sequences of HIV-1

cis-acting packaging sequences of HIV-1 are *cis*-acting sequences involved in packaging, and are located between the primer binding site and *gag* initiation site [Aldovini and Young, 1990; Clavel and Orenstein, 1990]. Running FOLDER on 10 *cis*-acting packaging sequences (*HIV*$_{NL43}$, *HIV*$_{LAI}$, *HIV*$_{HXB2R}$, *HIV*$_{MN}$, *HIV*$_{RCSF}$, *HIV*$_{SF2}$, *HIV*$_{NYCG}$, *HIV*$_{CDC4}$, *HIV*$_{HAN}$, *HIV*$_{RF}$) produces a covariation matrix with 145 potential helices after the covariation analysis phase. Figure 4.13 and
Figure 4.14: Another plausible structure for 10 HIV-1 cis-acting packaging sequences
4.14 show two plausible structures for 10 HIV-1 cis-acting packaging sequences inferred from the covariation matrix, which encompass a splice donor site (marked SD in the figures). The sequence and base numbering used in the figures are based on the HIV$_{NL43}$ genomic sequence. The first base of the figures is base 227 of the genomic sequence. They are the most stable structures conserved in the 10 HIV-1 cis-acting packaging sequences based on the default parameter values of Algorithm 4.4. Unlike TAR and RRE, the structure of the cis-acting packaging sequences have not been stabilized yet. As shown in Table 4.3, compensating base changes are rare in the helices due to little variation among sequences. In Table 4.3, the location of helices are based on the base numbering in the alignment of the 10 sequences, which includes gaps. The last sequence in each helix is a consensus sequence of the 10 sequences.

Figure 4.15 shows an optimal structure of the HIV$_{LAI}$ packaging sequence with minimum energy (-24.5 kcal/mol), predicted by FOLD [Zuker and Stiegler, 1981] in version 7.1 of the GCG Sequence Analysis Software Package running on a VAX/VMS version V5.4-2. Figure 4.16 is an optimal structure of the HIV$_{NL43}$ packaging sequence with minimum energy (-25.5 kcal/mol), predicted by the same program. The base sequences of the cis-acting packaging sequences of HIV$_{LAI}$ and HIV$_{NL43}$ are the same except at two bases (boxed in Figure 4.15 and 4.16) and are expected to have similar structures. Their structures predicted by FOLD, however, are radically different from each other. The two sequences are different at their first base (base 228 of their genomic sequences), but this difference did not affect the structural difference at all as shown in Figure 4.15 and 4.16. The actual and only reason for their structural difference is at the difference of base 281, which is A for HIV$_{LAI}$ and C for HIV$_{NL43}$. From the perspective of phylogenetic comparative analysis, the structure of Figure 4.15 is not as stable as that of Figure 4.16. This result shows that the structure predicted by the energy minimization method is overly sensitive to point mutation of a sequence. Testing
<table>
<thead>
<tr>
<th>organism</th>
<th>helix 64-68/73-77</th>
<th>helix 22-28/40-46</th>
<th>helix 17-20/50-53</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIVNL43</td>
<td>uugg/cgcca</td>
<td>cuug/cgccaag</td>
<td>cucc/cgagg</td>
</tr>
<tr>
<td>HVLAI</td>
<td>uugg/cgcca</td>
<td>cuug/cgccaag</td>
<td>cucc/cgagg</td>
</tr>
<tr>
<td>HIVHXB2R</td>
<td>uugg/cgcca</td>
<td>cuug/cgccaag</td>
<td>cucc/cgagg</td>
</tr>
<tr>
<td>HIVMN</td>
<td>uugg/cgcca</td>
<td>cuug/cgccaag</td>
<td>cucc/cgagg</td>
</tr>
<tr>
<td>HIVJRCSF</td>
<td>uggug/cgccg</td>
<td>cuug/cagcaag</td>
<td>cucc/cgagg</td>
</tr>
<tr>
<td>HIVSF2</td>
<td>uggug/cgcgg</td>
<td>cuug/cagcaag</td>
<td>cucc/cgagg</td>
</tr>
<tr>
<td>HIVNY5CG</td>
<td>uggug/cgcgg</td>
<td>cuug/cgccaag</td>
<td>cucc/cgagg</td>
</tr>
<tr>
<td>HIVDC4</td>
<td>uggug/cgcgg</td>
<td>cuug/cgccaag</td>
<td>cucc/cgagg</td>
</tr>
<tr>
<td>HIVHAN</td>
<td>uggug/cgcgg</td>
<td>cuug/cgccaag</td>
<td>cucc/cgagg</td>
</tr>
<tr>
<td>HIVRF</td>
<td>cggug/ccggc</td>
<td>cuug/cgccaag</td>
<td>cucc/cgagg</td>
</tr>
<tr>
<td>CONSENSUS</td>
<td>uGGUG/GCGGG</td>
<td>GUGGCG/GCGCAAG</td>
<td>UCGC/CAGCAG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>organism</th>
<th>helix 2-7/107-112</th>
<th>helix 90-94/99-103</th>
<th>helix 3-7/109-113</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIVNL43</td>
<td>ucucuc/gagaga</td>
<td>cuagc/gcuag</td>
<td>cuuc/gagag</td>
</tr>
<tr>
<td>HVLAI</td>
<td>ucucuc/gagaga</td>
<td>cuagc/gcuag</td>
<td>cuuc/gagag</td>
</tr>
<tr>
<td>HIVHXB2R</td>
<td>ucucuc/gagaga</td>
<td>cuagc/gcuag</td>
<td>cuuc/gagag</td>
</tr>
<tr>
<td>HIVMN</td>
<td>ucucuc/gagaga</td>
<td>cuagc/gcuag</td>
<td>cuuc/gagag</td>
</tr>
<tr>
<td>HIVJRCSF</td>
<td>ucucuc/gagaga</td>
<td>cuagc/gcuag</td>
<td>cuuc/gagag</td>
</tr>
<tr>
<td>HIVSF2</td>
<td>ucucuc/gagaga</td>
<td>cuagc/gcuag</td>
<td>cuuc/gagag</td>
</tr>
<tr>
<td>HIVNY5CG</td>
<td>ucucuc/gagaga</td>
<td>cuagc/gcuag</td>
<td>cuuc/gagag</td>
</tr>
<tr>
<td>HIVDC4</td>
<td>ucucuc/gagaga</td>
<td>cuagc/gcuag</td>
<td>cuuc/gagag</td>
</tr>
<tr>
<td>HIVHAN</td>
<td>ucucuc/gagaga</td>
<td>cuagc/gcuag</td>
<td>cuuc/gagag</td>
</tr>
<tr>
<td>HIVRF</td>
<td>ucucuc/gagaga</td>
<td>cuagc/gcuag</td>
<td>cuuc/gagag</td>
</tr>
<tr>
<td>CONSENSUS</td>
<td>UCUCUC/GAGAGA</td>
<td>CUAGC/GCUAG</td>
<td>CUUC/GAGAG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>organism</th>
<th>helix 10-12/59-61</th>
<th>helix 82-84/106-108</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIVNL43</td>
<td>cgc/gcg</td>
<td>uuu/gga</td>
</tr>
<tr>
<td>HVLAI</td>
<td>cgc/gcg</td>
<td>uuu/gga</td>
</tr>
<tr>
<td>HIVHXB2R</td>
<td>cgc/gcg</td>
<td>uuu/gga</td>
</tr>
<tr>
<td>HIVMN</td>
<td>cgc/gcg</td>
<td>uuc/gga</td>
</tr>
<tr>
<td>HIVJRCSF</td>
<td>cgc/gcg</td>
<td>uuu/gga</td>
</tr>
<tr>
<td>HIVSF2</td>
<td>cgc/gcg</td>
<td>uug/gga</td>
</tr>
<tr>
<td>HIVNY5CG</td>
<td>cgc/gcg</td>
<td>uuu/gga</td>
</tr>
<tr>
<td>HIVDC4</td>
<td>cgc/gcg</td>
<td>uuu/gga</td>
</tr>
<tr>
<td>HIVHAN</td>
<td>cgc/gcg</td>
<td>uuu/gga</td>
</tr>
<tr>
<td>HIVRF</td>
<td>cgc/gcg</td>
<td>uuu/gga</td>
</tr>
<tr>
<td>CONSENSUS</td>
<td>CGC/GCG</td>
<td>UUU/GGA</td>
</tr>
</tbody>
</table>

Table 4.3: Covariation analysis of the helices in Figures 4.13 and 4.14
Figure 4.15: An optimal structure of HIV_{LAI} packaging sequences (base 227 – base 335 of HIV_{LAI} genomic sequence) predicted by Zuker and Stiegler’s folding program [1981]. SD stands for splice donor site.

FOLD on HIV_{LAI} or HIV_{NL43} sequences with a slightly different range which encompasses the packaging sequences produces very different structures (data not shown), where not a single part of Figure 4.15 or 4.16 is preserved. This indicates that the structure predicted by the energy minimization is also sensitive to range which contains a sequence of interest.

Recently Harrison and Lever have published a structure of the HIV-1 cis-packaging sequences [Harrison and Lever, 1992], which is identical to the structure of Figure 4.16, except that it has an additional G-C pair in stem III. In the structure of Figure 4.16 and the structure predicted by Harrison and Lever, there is uncertainty around the multiple loop and some of the helices (e.g., UCG/CGA) around the multiple loop may not exist in that form. Another structure of HIV
Figure 4.16: An optimal structure of HIV$_{NL43}$ packaging sequences (base 227 – base 335 of HIV$_{NL43}$ genomic sequence) predicted by Zuker and Stiegler’s folding program

packaging sequences proposed by Hayashi et al. [1992], is not considered to be stable either in terms of phylogeny or free energy.

4.5 Experiments with Random Sequences

The program was also tested on random sequences to address whether the predicted common structures might occur by chance in random sequences with the same base composition and/or the same degree of similarity. Two types of random sequences were generated in a method described by Chan et al. [1991]. The first type of random sequences are independent random random sequences with the same base composition as natural RNA sequences. The second types are similar random sequences with the same base composition as well as the same degree of similarity as natural RNA sequences. In the first type, the bases were chosen independently using a random number generator with the only constraint that the expected base composition be equal to that of natural RNA sequences. In the second type, the base composition is also same as that of natural sequences, but in addition, the sequences were 80\% or 90\% similar to one another. The similar sequences were generated by exchanging random base pairs in a parental sequence until the required level of difference between the new sequence and the
parental sequence was reached. The parental sequence was obtained by random shuffling of natural RNA sequences.

Table 4.4 summarizes the number of potential helices with minimum 3 base pairings for the comparison of 2, 5, and 10 HIV-1 TAR, *cis*-acting packaging sequences, and RRE sequences. Sequences with suffix I represent independent random sequences. Sequences with suffix S represent random sequences with 90% similarity. The results are the average of 10 computer runs. Independent random sequences have much fewer potential helices than similar random sequences, which in turn have fewer potential helices than homologous RNA sequences. In general, the total number of potential helices common to random sequences decreases as more random sequences are compared. The reason that 10 similar *cis*-acting packaging sequences (PKG-S) has more helices than 5 PKG-S sequences is because the program allows 10% mismatches. It is also interesting that there is no potential helix found for the comparison of 10 independent random sequences although there are several in 10 similar random sequences. In addition, potential helices in random sequences are, on the average, shorter than those in homologous RNAs (Table 4.5). 10 sequences were used for each analysis. The results shown for random sequences in Table 4.5 are the mean of 10 computer runs.

4.6 Extension to Broader Class of Physical Systems

RNA is not the only biological system in which folding occurs. Proteins are also found in nature as complex three-dimensional structures instead of as linear sequences, and their shape determines their biological function. While RNAs are made up of 4 types of ribonucleotides, proteins are made up of 20 types of amino acids. Their structural elements also differ; the secondary structures of

---

4The result of the experiment with RRE is shown in Appendix C.2. The covariation matrix of 10 RRE sequences has 561 potential helices.
<table>
<thead>
<tr>
<th></th>
<th>number of sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>helix length</td>
</tr>
<tr>
<td>TAR-I</td>
<td>≥ 3</td>
</tr>
<tr>
<td>TAR-S</td>
<td>≥ 3</td>
</tr>
<tr>
<td>PKG-I</td>
<td>≥ 3</td>
</tr>
<tr>
<td>PKG-S</td>
<td>≥ 3</td>
</tr>
<tr>
<td>RRE-I</td>
<td>≥ 3</td>
</tr>
<tr>
<td>RRE-S</td>
<td>≥ 3</td>
</tr>
</tbody>
</table>

Table 4.4: Number of potential helices in the covariation analysis of random sequences

<table>
<thead>
<tr>
<th></th>
<th>potential helix size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sequence</td>
</tr>
<tr>
<td>TAR-S</td>
<td></td>
</tr>
<tr>
<td>TAR</td>
<td></td>
</tr>
<tr>
<td>PKG-S</td>
<td></td>
</tr>
<tr>
<td>PKG</td>
<td></td>
</tr>
<tr>
<td>RRE-S</td>
<td></td>
</tr>
<tr>
<td>RRE</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5: Lengths of potential helices in the covariation analysis of similar random sequences and homologous RNA sequences
RNA consist of double-stranded parts (i.e., helices) and several types of single-stranded parts (i.e., hairpin loops, bulge loops, internal loops, multiple loops), but protein secondary structures are described in terms of alpha-helices, beta-sheets, and so on.

It is an open problem whether the same method can be used for predicting RNA structures and protein structures. However, we conjecture that the basic approach of our method for predicting RNA secondary structures should be extensible to predict protein secondary structures, although not the tertiary structures. Proteins are categorized by families like RNAs. If there is a homology between two protein sequences, it is reasonable to expect them to have similar structures. Therefore, comparative analysis of homologous protein sequences to identify possible pairings can be done in a similar way as we do for RNAs. Constructing secondary structures from the comparative analysis can also be done similarly. However, the detailed way to identify possible pairings of amino acids or to construct secondary structures must be changed because the knowledge used to determine protein foldings is different.

Although the current modeling system of RNA structures cannot be directly used to predict protein structures, it can be easily applied to find disulfide bonds of protein structures, which occur between two amino acids containing SH radicals (such as cysteines) and form very strong bonds in folding structures.

The method can also be extended to other general problems which involve matching or compatibility between objects. Any relation of compatibility or conflict between objects is represented in a covariation matrix, and a searching algorithm operates on the matrix to find all the possible solutions which satisfy the relation.
4.7 Related Research

There currently exists no reliable and automatic method for predicting folded structures. Existing theoretical approaches have limitations in their use either because the approaches are computationally too expensive to be practical, the predicted structures by them are not stable or the approaches heavily rely on manual work. The existing approaches to RNA secondary structure prediction can be classified into two types: energy minimization and phylogenetic comparison. The energy minimization method uses thermodynamic estimates of structural stability to determine folding structures with minimum or near minimum free energy [Sankoff et al., 1983; Zaker and Stiegler, 1981; Zaker, 1989b]. Limitations of this method are partly due to the uncertainty of the underlying energy model. As shown in this chapter, the method is also overly sensitive both to point mutation and to range of a sequence.

The phylogenetic comparative method examines, usually manually, homologous sequences to identify potential helices, which maintain complementarity in the sequences. The current model of secondary structures of 16S and 23S of rRNA were obtained by Noller, Woese, and their collaborators with this method in conjunction with biochemical experiments [Noller and Woese, 1981; Noller, 1984]. Some of the Noller-Woese procedure lends itself to an explicit computer algorithm [Waterman et al., 1984; Waterman, 1988], but the algorithm depends on storing all the patterns of interest and has limitations to be applied to sequences with potentially large number of patterns.

The algorithm by Sankoff [1985] for simultaneously aligning and folding $M$ sequences of length $N$ requires $O(N^{3M})$ time and $O(N^{2M})$ space, and is therefore not practical either in terms of time or space. Recently a Macintosh program for use in phylogenetic comparative analysis has been made available by Brown
[1991], which shows the result of the analysis in a covariation matrix. However, secondary structure cannot be determined directly from the covariation matrix, especially when many competing helices are present in the matrix. Manual scrutiny of the matrix for possible secondary structures is a laborious task and can overlook possibilities.

Mironov, Kister and their colleagues have worked on RNA structure formation process [Mironov and Kister, 1986; Mironov and Kister, 1989; Mironov et al., 1985; Kister et al., 1992]. In their work, the folding process of RNA secondary structures is described by a Markov chain, and the kinetics of the structure formation is analyzed using the Monte Carlo method. Instead of modeling the folding process by a series of stable structures at intermediate stages during the RNA sequence growth or in the equilibrium states, they consider the time factors of structure transitions and predict the most probable structures at each point in time during transcription.

Like us, several researchers in AI have used approximation in problem solving. Weld [1986] describes a simulator which uses aggregation as an abstraction technique for a continuous process. His approach cannot be applied to simulate the folding process because the process depends on the constituent bases rather than on any regularity of the process, such as repeating cycles. Most research in qualitative kinematics has focused on predicting systems’ behavior from the description of the systems [Faltlings, 1987a; Forbus et al., 1991; Gelsey, 1991; Joskowicz, 1989]. A general constraint common in those systems is that objects cannot occupy the same space. The RNA folding problem has a similar constraint in the sense that a base cannot pair with more than one base. However, the current technique of qualitative kinematics seems inapplicable to the RNA folding problem.
4.8 Directions for Further Research

FOLDER has been tested on several different types of RNAs, which include tRNAs, 5S rRNAs, 16S rRNAs, TAR, RRE, and cis-acting packaging sequences of HIV-1. Running FOLDER on other types of RNAs using same or different parameter values would not require any change to the program.

FOLDER takes the alignment of homologous sequences as input, and the alignment is not changed during the problem solving process. To test FOLDER on different alignments of the same homologous sequences, we have to modify the alignment and run the program again on the updated alignment. A better way for doing this would be to allow a small shift of a sequence segment during the analysis phase of homologous sequences, which have the same effect of trying FOLDER with a slightly different alignment.

The current version of FOLDER considers secondary interactions only and predicts a secondary structure. It can be extended to incorporate tertiary interactions such as pseudo knots, which are base pairings between a loop and another part. Some of conflicting diagonals in the covariation matrix constitute pseudo knots.

We currently use a discrete model both for the secondary structure of RNA and for the structure formation process. For the secondary structure, a discrete model is more appropriate than a continuous model because there are no continuously varying parameters to consider in the model by its nature. The process of the structure formation is also modeled in a discrete model by taking “snapshots” of the structures at a series of intermediate stages. We might want to consider a continuous model for the process, which enables us to take into account the rate that a structure unwinds its helices to form a new helix as RNA grows.
4.9 Summary

The folding structure of RNA determines its biological function, and therefore predicting the folding structure is important for understanding RNA. This chapter has described a modeling system FOLDER to solve the folding problems of a biological system, RNA. FOLDER first analyzes multiple homologous RNA sequences to filter potential helices, searches the helices to construct structures from them, and then simulates the process of the structure formation over time. Unlike conventional methods, the modeling process of FOLDER is guided by several sources of knowledge. Main sources of knowledge of FOLDER are homologous RNA sequences and energy models. The knowledge from biochemical experiments or biological properties were not used to predict the structures presented in this chapter, but it can be incorporated into the modeling process whenever it is available. Having such prior knowledge would further prunes the search space of potential helices.

The most plausible structures conserved in multiple homologous sequences are produced in the same amount of time and space as those of the conventional models for a single sequence. FOLDER has been successfully tested on various types of RNAs. It is known that there is no reliable or automatic way of predicting a common folding even if the related sequences are already aligned and that one way to obtain a common folding is to fold each sequence separately using an ordinary folding program and to search for common structures [Zuker, 1989a]. The algorithm of FOLDER is the first one provably practical both in terms of time and space for finding folding structures of homologous RNAs. The domain knowledge and experimental data have been also formalized for RNA structure prediction, and a framework for combining knowledge from several different sources was developed. The program was actually used to predict potentially stable folding
structures of cis-acting packaging sequences of HIV-1, for which no generally accepted structures exist. **FOLDER** is the first qualitative reasoning approach of AI to solve RNA folding problem. It can be used as useful aids in designing biological experiments to determine more accurate folding structures. Experiments can be done to test some parts of a predicted structure, incorporating the results into the next prediction. Successive predictions and experiments thus lead to a more correctly determined folding structure.
Chapter 5

Conclusions

5.1 Summary

Automatically creating models of physical systems from fundamental domain knowledge is a challenging task. We have made important progress in automating the process of modeling the physical phenomena of motion and folding by developing new methods which use basic knowledge from physics laws, biological principles, and other sources. The methods have been implemented in working programs and tested in the domains of mechanical devices, sailboats, and RNA molecules. Experimental results show that the methods which use the fundamental, simple concepts in the domains can still automatically generate correct models to reason about several different types of physical systems involving motion or folding.

The main conclusions derived from the works on motion and folding can be summarized as follows:

First, the process of model-building and simulation of physical systems involving motion or folding is automated from basic knowledge of the domain. The modeling systems start with the most fundamental, simple concepts in the domains of mechanics and biology. Starting with the fundamental knowledge, the systems can still generate powerful models to predict complex motions or folding structures.

Second, both the model of motion and the model of folding structure are
constructed by putting relevant model fragments together, although they are of different types. The model fragments for motion contain a set of equations, but the model fragments for folding are structural elements (i.e., helices). Both the model of motion and the model of folding structure are built and simulated by methods with a common framework at high level: given a description of a problem involving a physical system, the methods analyze the problem, search for model fragments, construct a model with model fragments which are applicable to current situation and which are consistent with each other, and apply the model to solve the problem.

5.2 Review of Thesis Goals

There were two main goals of this thesis work:

1. Automating the process of model-building and simulation of physical systems for spatial problems, where objects are related to each other either geometrically or topologically to satisfy a set of constraints on the physical systems.

2. Making the modeling process widely applicable to many different physical systems involving either motion or folding so that common domain theories can be shared and reused instead of being duplicated.

We demonstrated that the first goal is achieved by showing the experimental results of testing our approach on physical systems involving either motion or folding. Our modeling systems start with the most fundamental, simple concepts in the domains of mechanics and biology instead of any pre-compiled knowledge. The systems then automatically reason from the fundamental knowledge and generate powerful models which can be used to predict complex motions or folding
structures. We showed that physical systems with multiple moving objects in arbitrary configurations can be automatically modeled and simulated by reasoning explicitly about vector quantities and moving frames of reference. We also demonstrated that modeling and simulation of RNA folding structures can be reliably automated by combination of approximation of the knowledge from homologous RNA sequences and biochemical experimental data on structural stability.

Evidence of the generality of our methods across different types of physical systems was demonstrated by the experimental results of testing the methods on the spring-block systems in a variety of configurations, the sailboats in fluids, and a composite object of rigid bodies. There are also many other types of physical systems which can be modeled with no or minor changes to the current methods, including multiple rigid bodies connected by springs (Section 3.6 discussed this issue in detail). This extensibility to wide class of physical systems is possible for several reasons. First, domain knowledge is represented with general model fragments in a purely declarative, algorithm-independent form, and instantiated later for particular situations, so that common knowledge can be reused and shared between different physical systems and within a physical system. Second, instead of using a special purpose method intended to handle a certain class of physical systems only, a general method is used to construct and simulate models; model fragments relevant to a physical system being studied are identified and composed to formulate a model, and the model is applied to solve a problem.

However, motion and folding were not modeled by a single program in the current implementation. Oracle and Folder have a common characteristic inference structure, but they are two separate programs. Could Folder be embedded in Oracle? We suspect that it is very difficult because:

1. When folding occurs in a physical system, the connections (not just contacts) of its components are changed dramatically. Therefore, modeling a
folding system can no longer assume a fixed connectivity of its parts and consequently a fixed shape of the system as modeling a moving system can.

2. Types of knowledge used in reasoning about the two phenomena are inherently different and cannot be shared. A central part of a model of complex motion is a set of mathematical equations specifying constraints over the values of variables. On the other hand, spatial constraints which should be satisfied by folding are about the topological properties of RNA structures, expressed in a more descriptive way (see the folding constraints in section 4.1), rather than by widely agreed equations such as physics laws.

3. The mathematical representation of a model of motion lends itself to being composed from pieces of partially specified equations by parameter substitution or shared variables. Models of folding do not have this kind of composability. The model fragments composed for the models of folding are not general equations instantiated for the particular problem, but specific structural elements (i.e., helices) which do not need further instantiation. Therefore, the detailed way of collecting and composing model fragments for models of folding is different from that for models of motion.

4. For the RNA secondary structure, we create a discrete model because there are no continuously varying parameters to consider in the model by its nature; either a hydrogen bond exists between two bases or it does not. For the motion of a mechanical system, we build a continuous model because a continuous model is more informative in understanding time-varying behavior of the system than a discrete model. Constructing a discrete model by a program which is intended to produce a continuous model does not seem to be easy.
Although folding was modeled and simulated by a separate program, we made the program general so as to be applicable to any types of RNAs. The program was successfully tested on different types of RNAs, covering many tRNAs, 5S rRNAs, 16S rRNAs, TAR, RRE, and cis-acting packaging sequences of HIV-1.

5.3 Contributions

In previous chapters, we described the ways in which previous methods for reasoning about physical systems were inadequate for modeling motion or folding:

- Most research in qualitative physics has focused on certain types of processes only, and does not provide a method for properly reasoning about complex motion or folding.

- Interactions between modeling and simulation are neglected and they are studied more or less independently by assuming the existence of their counterpart.

- Existing approaches to modeling folding have limitations in their use either because they are computationally too expensive to be practical, their predicted structures are not stable, or they heavily rely on manual work.

The work of this thesis filled in the gaps in the previous research and advanced the state of the art in modeling motion and folding.

5.3.1 Contributions of Oracle

Since the models of motion are used in reasoning about physical systems, constructing such models is clearly a problem of considerable interest for AI. This work also contributes the area of mechanics because motion is the central problem which mechanics deals with. Automated modeling and simulation of motion
are of great value for mechanics because these models are widely used but difficult to formulate by hand. Among the specific contributions of Oracle are the following:

- A new method was developed which solves several problems with existing AI modeling work on motion by: (1) declarative, algorithm-neutral representation of physics knowledge; (2) simultaneous handling of multiple equations (algebraic or differential, linear or nonlinear); and (3) explicit handling of vector quantities and frames of reference.

- A framework was developed for automating the model-building process for physical systems with multiple moving objects in arbitrary configurations in three dimension. It can help people better understand physical systems involving complex motion by relieving them of the burden of building hand-made models, which are likely to be error-prone and time-consuming.

- The modeling and simulation process was made general as to be reusable for many different physical systems if the behavior of the physical systems can be described by a set of ordinary differential equations.

- The method was implemented in a working program, and it seems to be a useful tool kit for generating models from a library of reusable model fragments for different types of physical systems involving complex motion.

- The epistemology of physical system modeling was studied. In particular, the process by which models are constructed and simulated, and the types of knowledge used in the process, as well as the epistemological adequacy of a modeling system for broader class of physical systems were examined.

- Oracle can facilitate the process of designing a new mechanical system or improving an existing mechanical system by generating a behavioral model
of a candidate design solution, which will be compared with the desired behavior.

- ORACLE can help diagnose and repair a mechanical system from recognizing an abnormal behavior deviated from the predicted behavior of the program.

5.3.2 Contributions of FOLDER

The folding structure of RNA determines its biological function, and therefore predicting the folding structure and its growth over time is important for understanding RNA. The specific contributions of FOLDER are as follows:

- The algorithm of FOLDER is the first one provably practical both in terms of time and space for finding common folding structures of homologous RNAs.

- The domain knowledge and experimental data were formalized for RNA structure prediction.

- A framework for combining knowledge from several different sources was developed.

- FOLDER was actually applied to predict potentially stable folding structures of cis-acting packaging sequences of HIV-1, for which no generally accepted structures yet exist.

- FOLDER is the first qualitative reasoning approach of AI to solve RNA folding problem.

- The method was implemented in a working program, and it can be used as useful aids in designing biological experiments to determine more accurate folding structures.
Appendix A

Model Fragments of ORACLE

This appendix contains the model fragments currently available in ORACLE, represented in the actual Maple syntax.

`Newton2 := table([`
  `pheno='Newton’s second law of motion',`
  `entities=[r=solid],`
  `variables=[f=r[net_force(t)],`
    `P(t)=r[momentum(t)]],`
  `assumptions=[ ],`
  `equations=[diff(P(t)[1],t)=f[1],`
    `diff(P(t)[2],t)=f[2],`
    `diff(P(t)[3],t)=f[3]]`
]);`

`Euler := table([`
  `pheno='Euler’s equations for the time-dependence of the angular velocity',`
  `entities=[b=rbbody],`
  `variables=[I=b[principal_moments_of_inertia],`
    `Omega(t)=b[ang_velocity(t)],`
    `T=b[net_torque(t)]],`
  `assumptions=[ ],`
  `equations=[`
    `I[1]*diff(Omega(t)[1],t)+(I[3]-I[2])*Omega(t)[2]*Omega(t)[3]=T[1],`
    `I[2]*diff(Omega(t)[2],t)+I[1]*Omega(t)[3]*Omega(t)[3]=T[2],`
    `I[3]*diff(Omega(t)[3],t)+I[2]*Omega(t)[2]*Omega(t)[2]=T[3]];
`
\[I[2] \cdot \text{diff}(\Omega(t)[2],t) + (I[1] - I[3]) \cdot \Omega(t)[3] \cdot \Omega(t)[1] = T[2],\]
\[I[3] \cdot \text{diff}(\Omega(t)[3],t) + (I[2] - I[1]) \cdot \Omega(t)[1] \cdot \Omega(t)[2] = T[3]\]

Springforce1 := table([
    pheno='spring force at end1',
    entities=[s=spring],
    variables=[fc=s[force_con],
                  b=s[damping_coeff],
                  E1(t)=s[end1(t)],
                  E2(t)=s[end2(t)],
                  r=s[rest_length],
                  f=s[force1(t)]],
    assumptions=[ ],
    equations=[
        f[1]=fc*(sqrt((E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+
                      (E2(t)[3]-E1(t)[3])^2)*r*(E2(t)[1]-E1(t)[1])/sqrt((
                      E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+(E2(t)[3]-
                      E1(t)[3])^2)+b*diff(E2(t)[1]-E1(t)[1],t),
        f[2]=fc*(sqrt((E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+
                      (E2(t)[3]-E1(t)[3])^2)*r*(E2(t)[2]-E1(t)[2])/sqrt((
                      E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+(E2(t)[3]-
                      E1(t)[3])^2)+b*diff(E2(t)[2]-E1(t)[2],t),
        f[3]=fc*(sqrt((E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+
                      (E2(t)[3]-E1(t)[3])^2)*r*(E2(t)[3]-E1(t)[3])/sqrt((
                      E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+(E2(t)[3]-
                      E1(t)[3])^2)+b*diff(E2(t)[3]-E1(t)[3],t)]
    ]):
Springforce2 := table([  
pheno='spring force at end2',
entities=[s=spring],
variables=[fc=s[force_const],
b=s[damping_coeff],
E1(t)=s[end1(t)],
E2(t)=s[end2(t)],
r=s[rest_length],
f=s[force2(t)]],
assumptions=[ ],
equations=[
f[1]=-fc*(sqrt((E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+(E2(t)[3]-E1(t)[3])^2-r)*(E2(t)[1]-E1(t)[1])/sqrt((E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+(E2(t)[3]-E1(t)[3])^2-r)*diff(E2(t)[1]-E1(t)[1],t),
f[2]=-fc*(sqrt((E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+(E2(t)[3]-E1(t)[3])^2-r)*(E2(t)[2]-E1(t)[2])/sqrt((E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+(E2(t)[3]-E1(t)[3])^2-r)*diff(E2(t)[2]-E1(t)[2],t),
f[3]=-fc*(sqrt((E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+(E2(t)[3]-E1(t)[3])^2-r)*(E2(t)[3]-E1(t)[3])/sqrt((E2(t)[1]-E1(t)[1])^2+(E2(t)[2]-E1(t)[2])^2+(E2(t)[3]-E1(t)[3])^2-r)*diff(E2(t)[3]-E1(t)[3],t)]
]);

FDrag := table([  
pheno='frictional drag force on an object moving in fluid',
entities=[s=physical_object, f=fluid],
variables=[D=D[s[fdrag(t)]]],
sd = s[rel_fluid_direction_vector(t)],
sv = s[rel_fluid_speed(t)],
Pa = s[parasitic_area],
rho = f[density],
assumptions = [],
equations = [D[1] = (1/2*Pa*rho*sv^2)*sd[1],
               D[2] = (1/2*Pa*rho*sv^2)*sd[2],
               D[3] = (1/2*Pa*rho*sv^2)*sd[3]]
]):

Lift := table ([
  pheno = 'lift and lift induced force on an object in fluid',
  entities = [s=physical_object, f=fluid],
  variables = [l= s[lift(t)],
               SL = s[lift_mag(t)],
               sd = s[rel_fluid_direction_vector(t)],
               sv = s[rel_fluid_speed(t)],
               pd = s[perp_rel_fluid_dir(t)],
               Ca = s[effective_capture_area],
               rho = f[density]],
  assumptions = [s[rel_fluid_speed(t)] > 0],
]):

Lift_at_zero_speed := table ([
  pheno = 'lift and lift induced force on an object in fluid',
  entities = [s=physical_object, f=fluid],
variables=[L=s[lift(t)],
            sv=s[rel_fluid_speed(t)]]],
assumptions=[s[rel_fluid_speed(t)] = 0],
equations=[L[1]=0,
            L[2]=0,
            L[3]=0]
]);

Opt_speed_ratio_x := table ([
    pheno='optimum trim variable for minimizing speed_ratio_x',
    entities=[s=sailboat],
    variables=[eta_s=s[sail_efficiency],
                eta_h=s[hull_efficiency],
                beta=s[apparent_course_angle],
                x=s[speed_ratio_x],
                Tau=s[trim_variable],
                kappa=s[hull_to_sail_capture_ratio]],
    assumptions=[ ].
equations={
    2*Tau*sin(beta)-(1/eta_s+Tau^2)*cos(beta)-kappa/eta_h/x^2
    -x^2/kappa*(Tau*cos(beta)+1/2*(1/eta_s+Tau^2)*sin(beta))^2=0,
    2*sin(beta)-2*Tau*cos(beta)-2*x^2/kappa*(Tau*cos(beta)
    +1/2*(1/eta_s+Tau^2)*sin(beta))*cos(beta)+Tau*sin(beta))=0}
]);
Appendix B

Entity Types of ORACLE

This appendix contains entity classes currently available in ORACLE, represented in the actual Maple syntax. The entity classes are organized in a tree as pictured in Figure B.1, which includes a set of entities used in the examples of Chapter 3. Properties of a super class are inherited by its subclasses. For example, properties of solid are inherited by rigid body.

![Diagram of entity types]

Figure B.1: Taxonomy of entities of ORACLE
### Each frame has a name, one slot that specifies the class of which
### it is a member of (specified by a AK0 slot for subclasses or an ISA
### slot for individual elements), and zero or more additional slots,
### each with its own value. Facets allowed in a slot are 'value',
### 'form', 'if_needed', and 'range'. Values in the 'value' facet can
### be numbers, strings, sets, lists, tables, arrays, or any other
### expressions. Values in the 'form' facet are usually vectors defining
### the components of the vectors. Values in the 'if_needed' facets are
### procedure calls. Values in the 'range' facet contain expected
### interval of the values if they are known. Both slots and facets can
### be added or modified, or removed from a frame after they have been
### created.

```plaintext
classical_base := table ([
    AK0=object
  ]);

primitive_object := table ([
    AK0=classical_base
  ]);

density := table ([
    AK0=primitive_object,
    mass(t)=[value=null],
    principal_moments_of_inertia(t)=[value=null,
      form=linalg[vector](3, [I1, I2, I3])],
    position(t)=[value=null,
      form=linalg[vector](3, [x(t), y(t), z(t)])],
    orientation(t)=[value=null,
```
form=linalg[vector](3, [phi(t),theta(t),psi(t)]),
velocity(t)=[value=null,
form=linalg[vector](3, [u(t),v(t),w(t)]),
if_needed=[derive_vel, position(t)],
ang_velocity(t)=[value=null,
form=linalg[vector](3, [omega1(t),omega2(t),omega3(t)]),
if_needed=[derive_ang_vel, orientation(t)],
momentum(t)=[value=null,
if_needed=[derive_momentum, mass(t), velocity(t)]],
ang_momentum(t)=[value=null,
if_needed=[derive_ang_momentum,
principal_moments_of_inertia(t), ang_velocity(t)]],
mf=[Newton2, Euler]
]):

rigidbody := table ([
AK0=solid,
mass(t)=[value=null,
if_added=[compute, momentum(t)]],
principal_moments_of_inertia(t)=[value=null,
form=linalg[vector](3, [I1,I2,I3])],
position(t)=[value=null,
form=linalg[vector](3, [x(t),y(t),z(t)]),
orientation(t)=[value=null,
form=linalg[vector](3, [phi(t),theta(t),psi(t)]),
velocity(t)=[value=null,
form=linalg[vector](3, [u(t),v(t),w(t)]),
if_needed=[derive_vel, position(t)],
ang_velocity(t)=[value=null,
form=linalg(vector)(3, [omega1(t), omega2(t), omega3(t)]),
if_needed=[derive_ang_vel, orientation(t)],
momentum(t)=[value=null,
if_needed=[derive_momentum, mass(t), velocity(t)],
ang_momentum(t)=[value=null,
if_needed=[derive_ang_momentum,
principal_moments_of_inertia(t), ang_velocity(t)],
force1_position_in_bf(t)=[value=null,
if_added=[compute, force1_position(t)],
force1_position(t)=[value=null,
if_needed=[derive_force_position, force1_position_in_bf(t)],
force2_position_in_bf(t)=[value=null,
if_added=[compute, force2_position(t)],
force2_position(t)=[value=null,
if_needed=[derive_force_position, force2_position_in_bf(t)],
force1(t)=[value=null],
force2(t)=[value=null],
net_force(t)=[value=null,
form=linalg(vector)(3, [F1(t), F2(t), F3(t)]),
if_added=[compute, net_torque(t)],
net_torque(t)=[value=null,
form=linalg(vector)(3, [T1(t), T2(t), T3(t)]),
if_needed=[derive_net_torque, force1_position_in_bf(t),
force1(t), force2_position_in_bf(t), force2(t)],
KE(t)=[value=null,
if_needed=[derive_KE, mass(t), velocity(t),
principal_moments_of_inertia(t), ang_velocity(t)],
PE(t)=[value=null],
mf=[Newton2, Euler]
rigidbody_consmass := copy(rigidbody):
rigidbody_consmass := subs(rigidbody=rigidbody_consmass,
    eval(rigidbody_consmass)):
rigidbody_consmass[AK0] := rigidbody:
rigidbody_consmass := subs({mass(t)=mass,
    principal_moments_of_inertia(t)=principal_moments_of_inertia},
    eval(rigidbody_consmass)):
temp := rigidbody_consmass[mass(t)]:
rigidbody_consmass[mass(t)] := evaln(rigidbody_consmass[mass(t)]):
rigidbody_consmass[mass] := temp:
temp := rigidbody_consmass[principal_moments_of_inertia(t)]:
rigidbody_consmass[principal_moments_of_inertia(t)] := evaln(
    rigidbody_consmass[principal_moments_of_inertia(t)]):
rigidbody_consmass[principal_moments_of_inertia] := temp:
rigidbody_consmass[KE(t)] := [value=null,
    if_needed=[derive_KE,mass,velocity(t),
        principal_moments_of_inertia, ang_velocity(t)]]:

block := copy(rigidbody_consmass):
block := subs(rigidbody_consmass=block, eval(block)):
block[AK0] := rigidbody_consmass:

disc := copy(rigidbody_consmass):
disc := subs(rigidbody_consmass=disc, eval(disc)):
disc[AK0] := rigidbody_consmass:
disc[radius] := [value=null]:
rod := copy(rigidbody consts\n\nmass);
rod := subs(rigidbody consts\n\nmass=rod, eval(rod));
rod[AK0] := rigidbody consts\n\nmass:

hull := copy(rigidbody consts\n\nmass):
hull := subs(rigidbody consts\n\nmass=hull, eval(hull)):
hull[AK0] := rigidbody consts\n\nmass:
hull[effective capture area] := [value=null, range=0..infinity];
hull[parasitic area] := [value=null, range=0..infinity];
hull[efficiency] := [value=null,
    if_needed=[derive efficiency, effective capture area,
    parasitic area]]:
hull[lift coeff] := [value=null];
hull[drag coeff] := [value=null];
hull[lift(t)] := [value=null,
    form=linalg[vector](3,[Lx(t),Ly(t),Lz(t)])]:
hull[fdrag(t)] := [value=null,
    form=linalg[vector](3,[FDx(t),FDy(t),FDz(t)])]:
hull[lag(t)] := [value=null, range=0..infinity];
hull[drag lag(t)] := [value=null, range=0..infinity];
hull[fluid] := [value=null];
hull[rel fluid speed(t)] := [value=null, range=0..infinity,
    if_needed=[derive rel fluid speed, fluid, speed(t), direction(t)]]:
hull[rel fluid direction(t)] := [value=null,
    if_needed=[derive rel fluid direction, fluid, speed(t),
    direction(t)]]:
hull[rel fluid direction vector(t)] := [value=null,
    if_needed=[derive rel fluid direction vector, fluid, speed(t),
    direction(t)]]:
hull[perp_rel_fluid_dir(t)] := [value=null,
if_needed=[derive_perp_rel_fluid_dir, rel_fluid_direction_vector(t),
fluid, speed(t), direction(t)]]:

hull[mf] := [FDrag, Lift, Lift_at_zero_speed]:

flexible_body := copy(solid):
flexible_body := subs(solid=flexible_body, eval(flexible_body)):
flexible_body[AK0] := solid:

spring := table ([$
AK0=flexible_body,
force_const=[value=null],
damping_coeff=[value=null],
rest_length=[value=null],
end1(t)=[value=null, form=linalg[vector](3,[e1x(t),e1y(t),e1z(t)]),
end2(t)=[value=null, form=linalg[vector](3,[e2x(t),e2y(t),e2z(t)]),
if_added=[compute, s_length(t), orientation(t)]]),
s_length(t)=[value=null,
if_needed=[derive_spring_length, end1(t), end2(t)]]),
position(t)=[value=null, form=linalg[vector](3,[x(t),y(t),z(t)]),
orIENTATION(t)=[value=null,
form=linalg[vector](3,[phi(t),theta(t),psi(t)]),
if_needed=[derive_spring_orient, end1(t), end2(t)]]),
force1_bf(t)=[value=null,
form=linalg[vector](3,[f1x(t),f1y(t),f1z(t)]),
force2_bf(t)=[value=null,
form=linalg[vector](3,[f2x(t),f2y(t),f2z(t)]),
force1(t)=[value=null, form=linalg[vector](3,[F1x(t),F1y(t),F1z(t)]),
force2(t)=[value=null, form=linalg[vector](3,[F2x(t),F2y(t),F2z(t)]),]
\[ PE(t) = [\text{value=null}, \]
\[ \text{if\_needed} = [\text{derive\_spring\_PE, force\_const, end1(t), end2(t),} \]
\[ \text{rest\_length}], \]
\[ \text{mf} = [\text{Springforce1, Springforce2}] \]
\]

\[
sail := \text{table}\left(\begin{array}{|l|}
\hline
\text{AK0=flexible\_body,} \\
\text{effective\_capture\_area=} [\text{value=null, range=0..infinity}], \\
\text{parasitic\_area=} [\text{value=null, range=0..infinity}], \\
\text{efficiency=} [\text{value=null, if\_needed} = [\text{derive\_efficiency,} \]
\text{effective\_capture\_area, parasitic\_area}], \\
\text{lift\_coeff=} [\text{value=null}], \\
\text{drag\_coeff=} [\text{value=null}], \\
\text{lift}(t) = [\text{value=null, form=linalg}[\text{vector}](3, [Lx(t), Ly(t), Lz(t)])], \\
\text{fdrag}(t) = [\text{value=null, form=linalg}[\text{vector}](3, [FDx(t), FDy(t), FDz(t)])], \\
\text{lift\_mag}(t) = [\text{value=null, range=0..infinity}], \\
\text{drag\_mag}(t) = [\text{value=null, range=0..infinity}], \\
\text{fluid=} [\text{value=null}], \\
\text{rel\_fluid\_speed}(t) = [\text{value=null, range=0..infinity,} \]
\text{if\_needed} = [\text{derive\_rel\_fluid\_speed, fluid, speed}(t), \]
\text{direction}(t)], \\
\text{rel\_fluid\_direction}(t) = [\text{value=null,} \]
\text{if\_needed} = [\text{derive\_rel\_fluid\_direction, fluid, speed}(t), \]
\text{direction}(t)], \\
\text{rel\_fluid\_direction\_vector}(t) = [\text{value=null,} \]
\text{if\_needed} = [\text{derive\_rel\_fluid\_direction\_vector, fluid, speed}(t), \]
\text{direction}(t)], \\
\text{perp\_rel\_fluid\_dir}(t) = [\text{value=null},
\hline
\end{array}\right)\]


if_needed=[derive_perp_rel_fluid_dir,
  rel_fluid_direction_vector(t), fluid, speed(t), direction(t)],
mf=[FDrag, Lift, Lift_at_zero_speed]
]);

fluid := table ([
  AK0=primitive_object,
  viscosity=[value=null],
  density=[value=null],
  kinematic_viscosity=[value=null,
    if_needed=[derive_k_viscosity, viscosity, density]],
  static_pressure=[value=null],
  speed_sound=[value=null],
  mass=[value=null, range=0..infinity],  # for Lagrangian framework
  volume=[value=null, range=0..infinity],  # for Eulerian framework
  steady=[value=null],  # boolean
  uniform=[value=null],  # boolean
  velocity_potential(t)=[value=null,
    if_needed=[derive_potential, velocity]],
  velocity(t)=[value=null,
    if_needed=[derive_vel, velocity_potential]],
  rotation(t)=[value=null, if_needed=[derive_rotation, velocity]],
  vorticity(t)=[value=null, if_needed=[derive_vorticity, velocity]],
  dynamic_pressure(t)=[value=null,
    if_needed=[derive_d_pressure, density, velocity]],
  velocity_field(t)=[value=null]
  Re=[value=null, if_needed=[derive_Re, length, velocity,
    kinematic_viscosity]],
  Fr=[value=null, if_needed=[derive_Fr, velocity, length]],
Mach=[value=null, if_needed=[derive_Mach, velocity, speed_sound]],
  mf=[],
]:

water := copy(fluid):
water := subs(fluid=water, eval(water)):
water[AK0] := fluid:
water[density] := [value=1.983]:
water[orientation(t)] := [value=null,
  form=linalg[vector](3, [phi(t), theta(t), psi(t)]),
  if_needed=[derive_fluid_orientation, direction(t)],
  if_added=[compute, direction(t)]]:
water[velocity(t)] := [value=null,
  form=linalg[vector](3, [u(t), v(t), w(t)]),
  if_needed=[derive_fluid_velocity, speed(t), direction(t)],
  if_added=[compute, speed(t)]]:
water[speed(t)] := [value=null, range=0..infinity,
  if_needed=[derive_speed, velocity(t)],
  if_added=[compute, velocity(t)]]:
water[direction(t)] := [value=null,
  if_needed=[derive_fluid_direction, orientation(t)],
  if_added=[compute, orientation(t)]]:

air := copy(fluid):
air := subs(fluid=air, eval(air)):
air[AK0] := fluid:
air[density] := [value=0.00238]:
air[orientation(t)] := [value=null,
  form=linalg[vector](3, [phi(t), theta(t), psi(t)]),
if_needed=[derive_fluid_orientation, direction(t)],
if_added=[compute, direction(t)]:

air[velocity(t)] := [value=null,
    form=linalg[vector](3, [u(t),v(t),w(t)]),
    if_needed=[derive_fluid_velocity, speed(t), direction(t)],
    if_added=[compute, speed(t)]:

air[speed(t)] := [value=null, range=0..infinity,
    if_needed=[derive_speed, velocity(t)],
    if_added=[compute, velocity(t)]:

air[direction(t)] := [value=null,
    if_needed=[derive_fluid_direction, orientation(t)],
    if_added=[compute, orientation(t)]:

composite_object := table ([
    AK0=physical_object,
    parts=[value=null],
    position(t)=[value=null,
        form=linalg[vector](3, [x(t),y(t),z(t)])],
    orientation(t)=[value=null,
        form=linalg[vector](3, [phi(t),psi(t),theta(t)])],
    velocity(t)=[value=null,
        form=linalg[vector](3, [u(t),v(t),w(t)]),
        if_needed=[derive_vel, position(t)],
    ang_velocity(t)=[value=null,
        form=linalg[vector](3, [omega1(t),omega2(t),omega3(t)]),
        if_needed=[derive_ang_vel, orientation(t)]],
    mf=[]
]);
ship := copy(composite_object):
sailboat := copy(ship):
sailboat := subs(ship=sailboat, eval(sailboat)):
sailboat[speed_ratio_x] := [value=null]:
sailboat[speed_ratio_y] := [value=null]:
sailboat[mf] := [Newton2, Opt_speed_ratio_xy]:

sloop := copy(sailboat):
sloop := subs(sailboat=sloop, eval(sloop)):
sloop[AK0] := sailboat:
Appendix C

Other Experimental Results of Folder

This chapter shows the experimental results of testing Folder on other types of RNAs.

C.1 Prokaryotic 5S rRNA

Folder has been experimented with 34 prokaryotic 5S ribosomal RNAs, whose alignment (shown in Figure C.1) was taken from [Waterman, 1988]. The result is similar to that of the experiment with tRNA in the sense that there are not many potential helices in the covariation matrix. There are only 5 potential helices in the covariation matrix of the 5S rRNA sequences (see Figure C.2 for the covariation matrix) and 4 of them constitute the normally accepted stems of prokaryotic 5S rRNA structure [Waterman, 1988; Fox and Woese, 1975; Singhal and Shaw, 1983; Göringer and Wagner, 1988] (Figure C.3). Base numbering of the structure in the figure is based on the first sequence of the alignment.

C.2 RRE of HIV-1

RRE (rev-responsive element) of HIV-1 is one of the cis-acting sequences involved in gene regulation, and lies in the env region of the HIV-1 genome [Malim et al., 1990; Dayton et al., 1992]. Comparative analysis of 10 HIV-1 RRE sequences (HIVNL43, HIVLAI, HIVHXB2R, HIVMN, HIVJRC5, HIVSF2, HIVNY55G, HIVCDC4, HIVHAN, HIVRF) results in a covariation matrix which has 561 potential helices.
Figure C.1: Alignment of 34 prokaryotic 5S rRNAs
Figure C.2: Covariation matrix of the 34 Prokaryotic 5S rRNAs, generated by FOLDER
Figure C.3: Common structure of the 34 Prokaryotic 5S rRNAs

The covariation matrix is too big to present here. Figure C.4 shows the common structure of 10 HIV-1 RRE sequences, which is more complex than the other cis-acting sequences of HIV-1. The first base of the figure is base 7323 of HIV$_{NL43}$ genomic sequence. The structure has 5 stems around the first multiple loop and 3 stems around the second multiple loop. The structure coincides with the published structure [Malim et al., 1990; Dayton et al., 1992] except for the stem IIb in the second multiple loop. From the perspective of the phylogeny, the corresponding stem of [Malim et al., 1990; Dayton et al., 1992] is not as stable as stem IIb of Figure 2C due to large variations of the sequences in that region, which do not conserve base pairings.

C.3 16S rRNA

16S ribosomal RNA is a very long molecule consisting of about 1500 bases. The experiment with 16S rRNAs was done by dividing the molecules into four folding domains because the current implementation of FOLDER has a limitation in its
Figure C.4: Common structure of 10 RRE sequences of HIV-1
input file format. Each sequence is assumed to be in a single line, which prohibits testing sequences larger than about 1K bases. Thus the 16S rRNA sequences were split into 4 domains (5’ domain, central domain, 3’ major domain, and 3’ minor domain) based on [Lewin, 1987], and tested separately. Ten Eubacteria and Archaeabacteria 16S rRNAs (E.coli, P.vulgaris, M.capricol, Hf.volcani, Hc.morrhua, Mc.vanniel, Sul.solfat, Hb.catirub, Hb.halobiu, Msp.hungat) were used for testing. The predicted structures of the four domains are quite satisfactory; all the helices with minimum 3 canonical or G-U pairs common to the individual 16S rRNA structures of [Noller, 1984; Guetell et al., 1985] have been identified; there are only three extra helices (126 Gga/Ucc 137 in central domain, 383 UucGG/cuGGA 421 and 429 GUA/Uac 460 in 3’ major domain. A consensus sequence is used in drawing the structure.
Figure C.5: Common structure of the 5' domain of 16S rRNAs
Figure C.6: Common structure of the central domain of 10 16S rRNAs
Figure C.7: Common structure of the 3’ major domain of 10 16S rRNAs
Figure C.8: Common structure of the 3' minor domain of 10 16S rRNAs
References


Vita

Kyungsook Han


1983 B.S., Seoul National University.


1985 M.S. in Computer Science, Korea Advanced Institute of Science of Technology.

1985-86 Member of Research Staff, ETRI, Korea.

1986-89 Graduate work in Computer Science, University of Minnesota, Minneapolis.

1989 M.S. in Computer Science, University of Minnesota, Minneapolis.

1989-94 Graduate work in Computer Science, Rutgers, The State University of New Jersey, New Jersey.


1994  Ph.D. in Computer Science.