Int J Adv Manuf Technol (2006) 28: 435–444
DOI 10.1007/s00170-004-2399-6

ORIGINAL ARTICLE

Y.L. Tian · H.J. Zou · W.Z. Guo

An integrated knowledge representation model for the computer-aided conceptual design of mechanisms

Received: 15 May 2004 / Accepted: 31 August 2004 / Published online: 4 May 2005

Abstract An integrated knowledge representation model, namely the topology structure behaviour function (TSBF) model, is presented for the computer-aided conceptual design of mechanisms (MCACD) in this paper. The model covers both qualitative and quantitative knowledge representations of generic mechanisms. A class hierarchy consists of the abstract mechanism, the embryo mechanism, and the concrete mechanism is then proposed for object-oriented modelling. Based on the TSBF model, several reasoning techniques are integrated to achieve a relatively comprehensive environment for MCACD. The corresponding reasoning process is mainly based on a backward chaining of solutions representation and retrieval, a forward chaining of compositional behaviour reasoning with constraint propagation and satisfaction, and a forward chaining of type synthesis. Coarse optimizations for certain mechanisms are also integrated on the quantitative level. The applicability of the new model is demonstrated by the conceptual design of a zigzag mechanism.

Keywords Conceptual design · Knowledge representation · Mechanism · Object-oriented modelling · Type synthesis

1 Introduction

In the engineering design process, conceptual design is a very important stage in which many critical elements of a product or scheme are determined [1]. It is also called design synthesis, which can be defined as the synthesis of new candidate design concepts [2]. Formalizing the conceptual design of mechanisms has raised much interest in both the mechanism community and the artificial intelligence (AI) community for decades. Nevertheless it remains a challenge to develop a general-purpose system for computer-aided conceptual design of mechanisms (MCACD). The difficulties mainly arise from the intrinsically complicated geometrical and topological characteristics in a mechanism that are normally absent in a general physical device. Kota et al [3] summarized the research on MCACD in depth about a decade ago. They stated that the conceptual design of mechanisms is a mixture of art and science, and suggested that several approaches are complementary. Since then, new progress has been made. Nevertheless, some later literature reviews (for instance [4]) only surveyed progress related to AI and regretfully neglected the effective research in the mechanism community. Therefore, an updated, brief review is worthy to be presented first.

In the mechanism community, there has been a long line of research considering number synthesis and type synthesis for the conceptual design of mechanism structures. In the mid-1960s, Freudenstein [5], the "father of modern kinematics", researched number synthesis to investigate the topology of mechanisms. Since then, a significant effort has been made to solve the number synthesis problem (see [6] for a comprehensive review). Meanwhile, various type synthesis methods employing number synthesis results to generate practical kinematic structures have been developed. Pioneering research includes Freudenstein's direct kinematic structure enumeration methods [5], and Johnson's associated linkage methods [2]. Recently, Tsai [7] extended Freudenstein's work and developed a complete type synthesis system that is especially powerful in the conceptual design of complex gear trains. Yan [8] presented a rule-based system for type synthesis using a generalization and regeneration process, and exploited more applications. Number synthesis and type synthesis are preferred for the creative and innovative design of mechanisms with time-invariant contacts, since they can enumerate topological structures exhaustively. However, the topological specifications and structural constraints required for a design task are not always available.

In the mechanism community, there has also been much research on MCACD employing AI techniques to integrate existing domain expertise. Many knowledge-based systems (KBS) and case-based reasoning (CBR) systems for mechanism selec-

Y.L. Tian (✉) · H.J. Zou · W.Z. Guo
School of Mechanical Engineering,
Shanghai Jiaotong University, Shanghai 200030, P.R. China
E-mail: tyl129@sjtu.edu.cn
Tel.: +86-21-62933054
Fax: +86-21-62932023

tion and pattern matching was developed in the early-1990s. Each system has typically been dedicated to a specific problem [3]. Nevertheless, not much progress on this aspect has been made since then.

On the other hand, MCACD research emphasising general knowledge representation and reasoning has become popular. Stahovich [9] offers AI research in the conceptual design of mechanical systems which can be broken into four categories: search, KBS, machine learning, and qualitative physical reasoning. However, his classification didn't distinguish the research on generic mechanisms from that on physical devices with simple mechanical components. In our opinion, the research considering the qualitative and semi-quantitative knowledge representation of existing mechanisms on kinematics and geometry, and research in related reasoning are fundamental works for the conceptual design of generic mechanisms. Joskowicz and Sacks [10] established a kinematic analysis algorithm which can automatically generate the input-output configuration space (C-space), and the transmission region diagram of a given mechanism providing its CAD information. Their representation system was estimated to cover about 60 percent of over 3500 mechanisms and kinematic pairs in Artobolevsky's mechanism compendia [11]. Later, Joskowicz and Neville [12] presented a more compact and expressive language in a Backus-Naur Form (BNF) to represent the qualitative behaviours of fixed axis mechanisms (including those with time-variant contacts). However, no algorithms for automated conceptual design were provided since the direct reasoning of a complicate mechanism is generally intractable. Therefore, many researchers have first tried to deal with the conceptual design of serially-connected mechanisms with time-invariant contacts only. Subramanian and Wang [13] presented a qualitative motion language and utilized constraint propagation for the combinational synthesis of serially-connected kinematic chains. Other forms of qualitative motion languages can be found in [14–17]. Among others, Chiou and Kota [16] made a successful attempt to establish an automated conceptual design system of serially-connected mechanisms. They identified 43 generic kinematic building blocks from several famous mechanism sourcebooks, and introduced a multi-levelled qualitative motion transformation matrix (MTM) to represent these mechanism blocks. Task decomposition becomes much easier by utilizing basic matrix manipulations. More recently, Ye et al. [17] applied data standardization technology to improve the efficiency of mechanism selections and kinematic behaviours reasoning. Meanwhile interesting research towards the conceptual design of mechanisms with time-variant contacts has also been carried out [18–22]. Most efforts are related to the qualitative configuration space (QC-space) theory. Nevertheless, currently, those approaches can only deal with very simple devices.

With qualified behaviour representations, comprehensive knowledge representation models of mechanisms can be then established. Kannapan and Marshek [14] presented a structure behaviour function (SBF) model. However, the definitions in their models are descriptive rather than formal. A SBF model with formal definitions of structure, behaviour and function for representing mechanisms, mainly based on C-space theory, was first suggested by Subramanian and Wang [13]. But their model was limited to represent simple open-looped kinematic chains. More recently, Roy et al. [23] presented a similar but more detailed model using the UML language attempting to deal with both the conceptual design and parameter optimization of generic mechanisms. Their model was not well organized, however, and its applications are limited to simple kinematic chains. It should be noted that all these models didn't explicitly consider the topological aspect of mechanisms.

In this paper, we extend our previous research of qualitative reasoning in kinematic behaviours [17] by integrating a topological reasoning. Meanwhile, the geometry reasoning interface is also considered. An integrated knowledge representation model called topology structure behaviour function (TSBF) model is presented for combined reasoning. The TSBF model has an additional topology level that is able to provide topological reasoning. Based on the TSBF model, a MCACD system capable of both qualitative behaviour reasoning and topological reasoning have been developed. Object-oriented modelling techniques are employed to establish the knowledge base and the reasoning process. Moreover, a mechanism simulator is integrated in the system in order to provide an early assessment of the obtained solutions. The system is also intended to extend the applicability of some existing software for conceptual design of mechatronic systems (e.g. the Schemebuilder [24]) that are quite weak in terms of the conceptual design of mechanism subsystems.

This paper is organized as follows: in Sect. 2, the conceptual design process model and the architecture of the MCACD system are illustrated; in Sect. 3, the TSBF knowledge representation model is defined and the object-oriented representation scheme for case representation is presented; in Sect. 4, the details of the reasoning process are presented; in Sect. 5, a design example is presented; and in Sect. 6, we offer our conclusions.

2 The MCACD system

2.1 Integrated conceptual design process model

A conceptual design process model describes when and how a reasoning mechanism works in a conceptual design process. A proper conceptual design process model is necessary to identify the required knowledge representations and to guide the development of a MCACD system. There are typically four design paradigms in the conceptual design process of mechanisms: forward chaining, backward chaining, step-wise refinement, and constraint propagation [1]. Generally, the first two are most relevant: forward chaining reasoning starts with problem specifications and proceeds forwards through intermediate states towards a solution. In contrast, backward chaining reasoning starts with a finite set of predefined solutions and proceeds backwards though intermediate states towards the initial specifications. The primary advantage of a forward chaining strategy is its ability to exhaustively enumerate solutions in a systematic and unbiased manner. However, the enumeration could lead to a prob-

lem of combinatorial explosion. The backward chaining strategy tends to mimic the typical reasoning steps of human designers. Domain experts must provide the solutions of many specific problems beforehand, which guarantees that there is a certain possibility for a given problem to fall under these solutions. In practice, type selection and pattern matching are the most commonly applied backward chaining techniques in the mechanism community [1, 2, 25–27]. However, type selection and pattern matching only consider the steps of case representation and retrieval in CBR. Though mechanism experts have also provided some redesign strategies for specific simple mechanisms, such as rewrite-rules for the redesign of couplings between adjacent pivots [14], rules for adaptive redesign are still lacking in the literature of mechanism conceptual design.

In view of the advantages and disadvantages of forward chaining and backward chaining, some researchers have adopted both chaining strategies in knowledge-based systems. Typical examples are rule-based systems for the compositional reasoning of compiled kinematic building blocks. However, the kinematic building blocks are always fixed and only simple solutions will be produced. Therefore, it is desirable to integrate a topological enumeration routine as another forward chaining reasoning to generate candidate sub-structures with new topologies. The new conceptual design process model is illustrated in Fig. 1. In the design process model, a CBR subsystem (located on the top) assists a designer in retrieving information and prototypes. The principal part is a KBS, namely a rule-based system for compositional reasoning. It takes motion specifications and topological specifications as input, and can recognize constraints. KBS solves the constraints by constraints propagation and satisfaction. There is a systematic type synthesis routine which acts as a topological forward chaining, and it offers transformational rules in the principal KBS through the generalization and regeneration of obtained sub-structures. In Fig. 1, the procedure of the forward chaining of type synthesis are marked by hollowed arrows. The type synthesis routine can also run independently. Details will be discussed in Sect. 4.

2.2 Architecture of the MCACD system

The architecture of a MCACD system, with respect to the proposed conceptual design process, is shown in Fig. 2. There are four parts to the system: a main program, an inference program, a simulation program, and an external database. The main program has three modules: a console module, a module for information searching and editing, and a module for mechanism evaluation and coarse optimization design. The inference program was developed using a professional logic program devel-

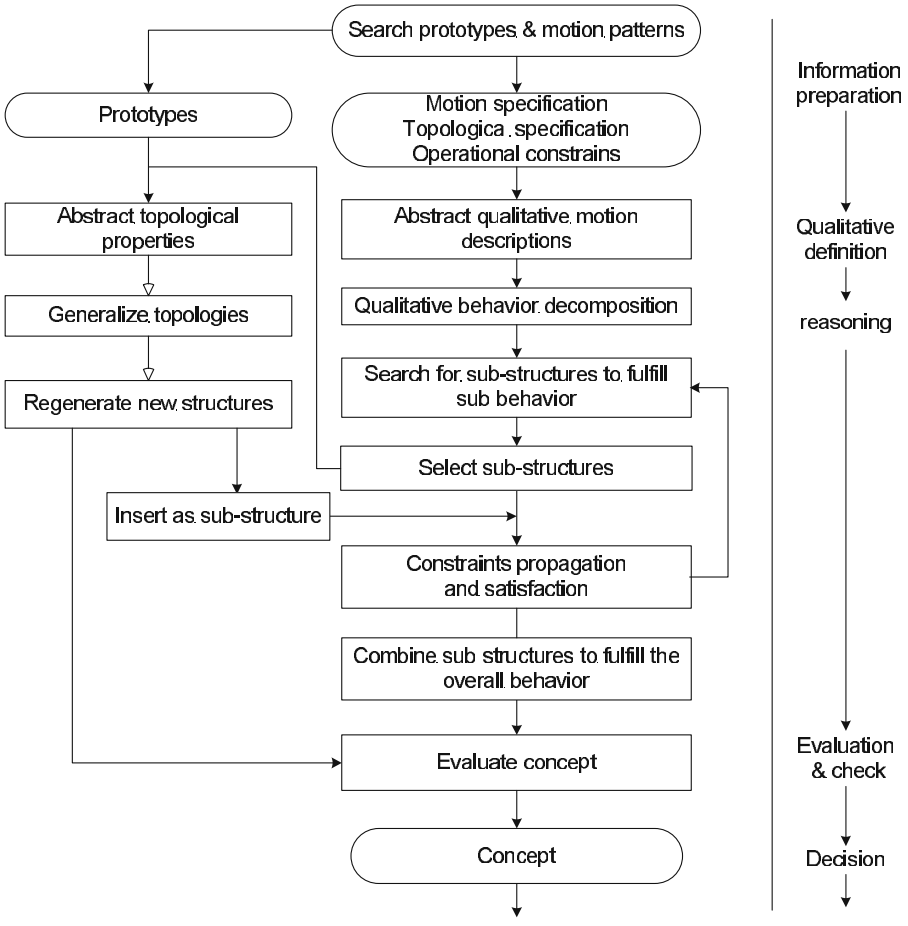


Fig. 1. Process model of mechanism conceptual design

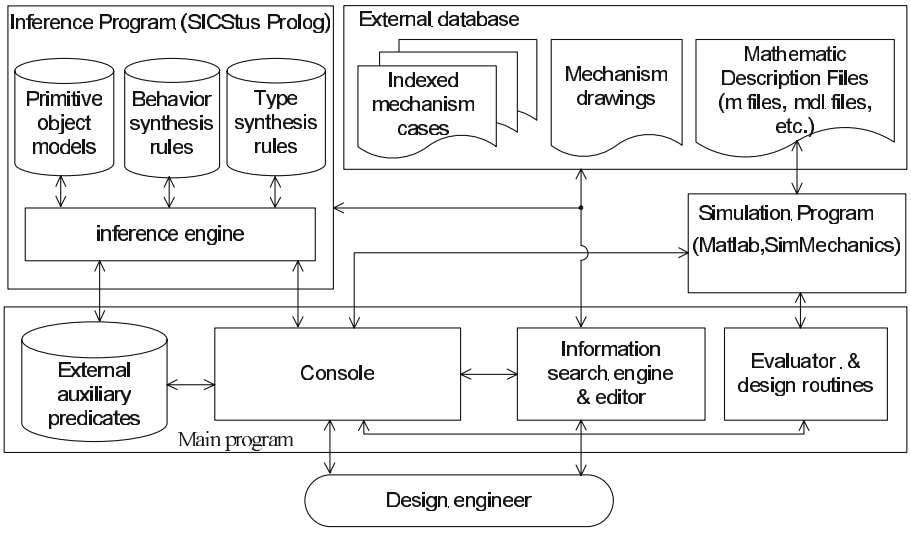


Fig. 2. Architecture of the MCACD system

oping system, SICStus Prolog [28]. The details of the reasoning mechanism will be discussed in later sections. The simulation engine was built on Matlab and SimMechanics (products of MathWorks Corporation). SimMechanics is a toolbox in Matlab for the kinematic and dynamic simulation of mechanisms. The main program can automatically generate and manipulate the SimMechanics models of most mechanism concepts, and can produce schematic sketches of mechanisms employing Matlab scripts and Matlab external programming interfaces [29]. The external database includes a number of mechanism catalogues with various indexes (e.g. usage, kinematic pairs, motion behaviours and curve patterns) for case retrieval. Moreover, mechanism drawings, qualitative motion descriptions and abstracted topological descriptions have been provided for some mechanisms in the external database. The operational details can be briefly introduced as follows:

1. A designer first uses the search engine to find desirable mechanism prototypes or curve patterns from the external database. Given the prototypes and related information, the designer edits the desired qualitative motion specifications and abstracted topological requirements. Other requirements can be input as heuristics to limit the search scope in the inference process.
2. These specifications and requirements are then parsed by the console module to construct a number of prolog predicates for reasoning. The console then loads the predicates into the SICStus prolog and runs the program to query solutions. The SICStus Prolog program collaborates with the console to complete the conceptual design process. The console also interacts with the designer by asking for selections during the conceptual design process.
3. When a concept is obtained, the console constructs its mathematic model file and then passes it to the Matlab simulation engine for simulation.
4. The simulation results are then evaluated by the evaluator module where some fast coarse-fine optimization routines have been provided to refine the design of specific mechanisms, such as dwell linkages. The designer can also provide his or her own optimization algorithms in Matlab scripts.

3 TSBF knowledge representation model

3.1 Formal definitions

To achieve a relatively comprehensive MCACD system, the establishment of a proper and compact model for both knowledge representation and the reasoning of mechanisms is fundamental. Formal definitions of the elements in a knowledge representation model are in turn the foundation for constructing the model. Informal definitions generally leads to unqualified models. For instance, the term function is semantically overloaded having both operational and purposive meanings as a relational concept. Consequently, considerable confusion exists in the literature about the use of the terms “function” and “behaviour”, and the confusion hinders clarifying the related reasoning. To express the TSBF model explicitly, a number of formal definitions and necessary explanations are given as follows:

- Configuration of a body: a configuration of a body B is defined as an isolated image point $\boldsymbol{x}$ in a n dimensional image space, with its state variables x_i, $i = 1, \ldots, n$ as the coordinates. Typical state variables include displacement and orientation for rigid bodies, and dimensions for elastic bodies. The configuration space of body B is formed by all its possible configurations. A configuration is generally dependent on the choice of reference frame, while the configuration space is generally frame-independent.
- Motion: the motion of a body is a function of time to its configuration space; in other words, the motion describes the changes of configuration with respect to time. The function is first order continuous but not necessary first order differentiable. A motion is intrinsically a function vector and can be represented by a screw, a quaternion, or a D-H matrix.

- Configuration of a mechanism: a configuration of a mechanism M consists of: (1) an aggregation of all the simultaneous body configurations of its members with all kinematic and geometrical constraints satisfied; and (2) contact states between these members. The configuration space of M is formed by all its possible configurations. In practice, generally only the contact states and the body configurations of the input members and output members are important; thus, the configuration space can be reduced to a simplified form, such as the input-output C-space [10].
- Quantitative behaviour: the quantitative behaviour of a mechanism is a function of time to its configuration space. The term "behaviour" is used because it implicitly indicates a function mapping from the input motions to the output motions. Note that the behaviour of a mechanism may be first order discontinuous, since the contact states are discrete variables. In others words, the mechanism can be a so-called "multiple-state mechanism", or a mechanism with time-variant contacts.
- Qualitative behaviour: there are two levels of qualitative behaviour. In the first level, C-space representation is replaced by QC-space representation, or all design variables in the quantitative behaviour equations are replaced by symbols. In the second level, only descriptive language (e.g. the first-order predicates logic) is used. Besides descriptions for the basic motion characteristics which decide the types of coordinates of C-space, other descriptions are called "operational constraints", which abstract the operational characteristics of the C-space with respect to time.
- Structure: structure is the concrete form of a mechanism. The structure of a mechanism can be presented as a pair (K, C), where K is the kinematic diagram of the mechanism and C is a set of geometric and kinematic constraints on K. The kinematic diagram K is often represented by a link adjacency matrix with specified kinematic pairs.
- Topology: the topology of a mechanism is an abstraction of its kinematic diagram. There are three levels of abstractions: the first level includes the global relationships between substructures, if the structure is decomposable; the second level includes the generalized kinematic chains without ground links and with only generalized revolute joints; and the third level includes the topological graphs. The first level is suitable for constructing a compound mechanism; the second level is suitable for type synthesis; and the third level is suitable for number synthesis.
- Function: generally, function relates to how the mechanism interacts with the environment. It is frequently expressed as a purpose predicate. However, a number of kinematicians argued that purpose is not an intrinsic property of a mechanism, and a mechanism can interact with the environment by arbitrary means. On the other hand, a sub-mechanism in a large mechanism may not interact with the environment, but it still has an intrinsic purpose, such as providing a required motion transformation. Nevertheless, purposes or usages are necessary since it is not always able to express explicitly the abstract operational characteristics. Thus, in our opinion, the functions of a mechanism can be its abstracted purposes or most abstract operational characteristics. While this definition is not formal, the domain is a set of well-defined function vocabularies.

Based on the above definitions, different levels of abstractions of a mechanism can be established. Namely, these are concrete mechanisms, abstract mechanisms, and embryo mechanisms. The multi-level representation of a mechanism will be much more efficient for qualitative reasoning. The architecture is also suitable to be represented by object-oriented languages.

- Concrete mechanism: a concrete mechanism is a physical object which includes four aspects: function, behaviour, structure and topology. Its name can be a string or a mechanism ID number. Additional properties are useful to assess its applicability, such as its advantages, disadvantages, efficiency, dexterity index, and limitations.
- Abstract mechanism: an abstract mechanism is an abstracted object with two aspects: function and qualitative behaviour. The qualitative behaviour normally includes qualitative input motions, qualitative output motions, and their relationship. It should include at least qualitative output motions.
- Embryo mechanism: an embryo mechanism is an abstracted object with generalized topological properties. It includes topology information and links to type enumeration strategies for generation of structure.

The object hierarchy features multiple layers: an abstract (embryo) mechanism can be a super class of abstract (embryo) mechanisms; an abstracted mechanism object can be a super class of concrete mechanism objects with equivalent qualitative behaviours and functions; and an embryo mechanism object can be a super class of concrete mechanism classes with equivalent topological properties. A concrete mechanism can inherit properties and methods from several abstract mechanisms and embryo mechanisms. Finally, coarse or fine parametric design procedures should be provided for a concrete mechanism whenever possible.

3.2 Qualitative representation

To represent the TSBF model in detailed forms, qualitative vocabularies and symbolic equations are necessary to be abstracted from applications of concrete mechanisms. Table 1 illustrate a number of qualitative vocabularies. Some details are omitted due to the space limits.

Based on the above qualitative vocabularies, definitions in the TSBF model can be represented in an extended Backus-Naur form (BNF). For example, a stacked double input–single output (DISO) concrete mechanism can be represented as shown in Table 2.

In the extended BNF description, symbols enclosed by angle brackets are non-terminals. Bold symbols are terminals. Capital symbols stand for terminals defined in a separate dictionary or catalogue. Other symbols (e.g. Qc-space, Symbolic-equation) indicate items from a non-finite domain. The suffix "+" is an abbreviation for one or more of an item. The suffix "2" indicates two

Table 1. Some qualitative descriptions

function	Operational characteristics and physical usage: rigid body guidance, function generation, general path generation, straight line path generation, axis transformation, enlarging motion range, indexing; escapement, differential, positioning, increase force, coupling, clamping, speed conversion, feeding, etc.
behaviour	Quantitative behaviour: C-space, equations, transmission region diagrams Qualitative behaviour: level 1: Qc-space, qualitative timing curves, symbolic equations level 2: Motion transformation classified by a triplet: (input-motion, output-motion, operational-constraints) Motion: rotation, translation, planar path (15 basic groups), planar motion, helix, spherical path, spatial path, etc. Operational constraints: axis direction, axis-transformation (six types), linear, nonlinear, continuous, intermittent, interchangeable, one-way driving, oscillation, dwell, double dwell, etc.
structure	Model files, link adjacency matrix, geometrical constraints, kinematic properties, geometrical properties, dimensions, degree of freedom, etc.
topology	Level 1: global layout type serial-chain, parallel (parallel-serial, serial-parallel), compound (11 groups) Level 2: generalized kinematic chain Level 3: line graph, contracted line graph

Table 2. EBNF description of DISO mechanism

<DSOMech>	::=	<Function><DISOBehaviour> <DISOStructure><DISOTopology>
<Function>	::=	<Function-Name>$^{+}$
<Function-Name>	::=	FUNCTIONVOCABULARY
<DISOBehaviour>	::=	<Input-motionL3>^{2}<Output-motionL3> <ConstraintsL3>$^{+}$
<ConstraintsL3>	::=	<MotionConstraintsL3> | <TransformationConstraintsL3>
<Transformation ConstraintsL3> ···	::=	<Degree-of-Freedom>, <Axes-Relationships>
<DISOTopology>	::=	<TopologyL1>
<TopologyL1>	::=	(<SISOMech>, <SISOMech>, <LayoutTypeID>)
<SISOMech>	::=	<UnitMech>| <ChainMech>
<ChainMech>	::=	<UnitMech>then<ChainMech>
<UnitMech>	::=	<Name><Function><UnitBehaviour> <UnitStructure><UnitTopology>
<UnitBehaviour>	::=	<UnitBhrL1><UnitBhrL2><UnitBhrL3>
< UnitBhrL3>	::=	<Input-motionL3><Output-motionL3> <OperationalConstraintsL3>$^{+}$
<Input-motionL3>	::=	rotation|translation|planarpath|planar motion|helix|sphericalpath|spatialpath
< UnitBhrL2>	::=	Qc-space, Qualitative-Timing-Curve, Symbolic-equation+
...		

occurrences. Compared to the BNF syntax for representing fixed axis mechanisms presented by Joskowicz and Neville [12], the syntax of the TSBF model has more specifications and can also represent compound mechanisms explicitly. However, though BNF is quite useful for specifying the syntax of mechanism representation models, utilizing an executable language rather than developing a new language should be considered to realize the syntax. The choice of a suitable executable language is discussed in next subsection.

3.3 Object-oriented representation

Plain prolog languages with the first-order predicate logic are widely used for knowledge representation and logic programming. Recently, many new features have been introduced to extend the capability of prolog language, such as higher-order logic and object-oriented programming. SICStus Prolog provides object-oriented logic programming grammar for modelling a large knowledge system with complex structures and relationships. In SICStus Prolog, an "object" is a collection of predicate definitions. Objects can be defined in a file, or dynamically created during the execution of a program. Moreover, the system automatically maintains a table of the object hierarchies and tables of class-instances. These characteristics afford greater flexibility for knowledge modelling and management. On the other hand, SICStus Prolog provides extended logical programming capability, including constraint logic programming (CLP) and constraint handling rules (CHR). And it supports external predicates written in C language.

Since object-oriented prolog is ideal for documenting both logic programming and object-oriented programming, we prefer to illustrate the modelling of mechanisms using the object-oriented prolog syntax provided by SICStus Prolog. We assume the reader is familiar with basic prolog predicates and the mode specification of prolog predicates.

A general class of object is defined as:

```
object class-identifier :: {
   sentence 1.    % Class hierarchy
   ...
   sentence i.    % Facts
   ...
   sentence m.    % Methods
   ...
}.
```

We first illustrate the class definitions of a single input–single output (SISO) abstract mechanism. In practise, elements of the TSBF model are classified and encapsulated as importable predicate modules for clear code reuse. Here, we put those predicates together for clarity.

```
abstractsisomechclass_1 :: {        % abstract class
  super(abstractmechclass).         % class hierarchy
  :- public(cousinabtractclass_/1). % public data
  :- public(rankedchildlist_/1).
  :- private(functionattr_/2).      % private data
  :- private(outputmotionattr_/2).  % level 3 output motion
  ...
  :- initialization(init/0).        % method protocols
  :- public(set_functionattr/2).
  :- public(get_functionattr/2)
  :- public(retractrankedsub/1).
  :- public(getnextcousin/1).
  ...
  init :-                           % method implementations
  predicates_for_initialization.
...
}
```

For SISO mechanisms, we also abstract several embryo mechanism classes.

```
embryosisomechclass_1 :: {          % abstract class
  super(embryomechclass).           % class hierarchy
  :- :use_module(structure).        % import modules
  :- public(cousinembryoclass_/1). % data
  :- private(topologicattr_/2).
  :- private(isbkc_/1).
  ...
  :- public(typesynthesis/2).       % method protocols
  ...
  typesynthesis (Structure, Constraintlist) :-% method
                                              % implementations
    isbkc(BKC),
    :getnewstructure(BKC, Structure),
    :satisfyconstraint(Structure, Constraintlist).
    ...
  }
```

The bottom level of the class hierarchies consists of all instantiable concrete mechanism prototypes. A prototype inherits predicates from its parent classes and also extends its parent classes with specific attributes and strategies. Most prototypes can be discretized precisely or approximately into a small number of instances for instance selection. For prototypes requiring a large number of instances, design strategies are preferred for coarse parametric selection. A prototype is demonstrated as:

```
concretesisomechproto_1 :: {        % instantiable class
  super(abstractsisomechclass_1). % class hierarchy
  super(abstractsisomechclass_m).
  super(embryosisomechclass_1).
  super(embryosisomechclass_n).
  ...
  % extended attributes
  :- public(mechanism_id_/1).
  :- public(cousinprotolist_/1).
  :- public(hasdesignstrategy_/2).
  :- private(inputexpression_/1).  % level 2 expressions
  :- private(outputexpression_/1).
  :- private(constraintexpression_/1).
  :- private(structureattr_/2).
  :- private(structureparam_/2).
  :- private(topologyattr_/2).
  ...
  :- initialization(init/0).         % method protocols
  :- public(findinstance/1).
  :- public(regeneratestructure/2).
  :- public(displayprotoinfo/0).
  :- public(generatemdlfile/1).      % external predicate
  :- public(applydesignstrategy/1). % external predicate
  :- public(set_structureparam/2).
  :- public(insertconstraints/0).
  :- public(retractconstraints/0).
  ...
  init :-                           % method implementations
  set_inputmotionattr(type, rotate),
  set_inputmotionattr(raxis, (0, 0, 1)),
  set_inputmotionattr(sense, ccw),
  set_outputmotionattr(continuous, reciprocating),
  ...
  regeneratestructure(Structure, Constraintlist) :-
  super(Super),
  Super::typesynthesis (Structure, Constraintlist),
  hasdesignstrategy(function_id,Strategy),
  applydesignstrategy(Structure, Strategy).
   ...
}
```

Mechanism instances can be created either statically or dynamically. Dynamic creation is preferred for creating and storing instances, since most properties of its prototype are inherited. The dynamic creation of a crank-rocker mechanism is given as follows:

```
% create a new instance
instance(crankrockerinstance_1, crankrockerproto).
% specialize the prototype
crankrockerinstance_1::set_structureparam(linklength1,0.4).
crankrockerinstance_1::set_structureparam(linklength3,1.5).
crankrockerinstance_1::set_outputmotionattr(extent, 50).
...
```

4 Reasoning in MCACD

The TSBF model and its multiple-layer, object-oriented representation provide a clear architecture for integrated reasoning in MCACD. We have extended the behavioural reasoning demonstrated in our previous research [17], and implemented topological reasoning in the SICStus Prolog system. We do not apply geometrical reasoning, but it can be realized as design strategies using external predicates for specific mechanism prototypes. The reasoning mechanism of the prolog language is basically backtracking and unification. The reasoning mechanism is simple but powerful. In our research, the abstraction of mechanisms and various constraints are adopted to improve the efficiency of plain backtracking.

4.1 Behaviour reasoning of mechanisms

There are typically three types of mechanisms requiring behavioural reasoning: single input–single output (SISO) mechanisms, single input–multiple outputs (SIMO) mechanisms, and multiple inputs compound (MIC) mechanisms. Synthesis of SISO mechanism is a fundamental reasoning task. The output motion-oriented behaviour decomposition is preferred since the types of output motions are more specific than that of input motions. The synthesis algorithm can be expressed as follows:

SISO_Syn(*inputmotion*, *outputmotion*, *constraintlist*, *Mech*)

1. If “inputmotion” = “outputmotion”, then “Mech” is unified with a utility machine and returned. Else, select an abstract mechanism “AM” with “AM::outputmotionattr” contains the principal motion properties (including operational constraints) of the required output motion “outputmotion”.
2. If no “AM” is found, then the task fails. Else, “AM” selects a child “CM” using a built-in predicate “sub/1” of the object-oriented prolog.
3. If “CM::inputmotionattr” contains the principal motion properties of the required“inputmotion”, and corresponding constraints in “constraintlist” are satisfied, then “CM” is inserted to a candidate mechanisms list. If the query is set as non-terminal, then the algorithm continues to find new candidates.
4. If “AM” can’t find any child that suits the requirements, then the design task is decomposed into:

```
AM::sub(CM),
SISO_Syn(inputmotion, CM::inputmotionattr, constraints,
     Mech1),
append(Mech1, CM, Mech).
```

5. The above qualitative reasoning generally feedbacks a large number of solutions even with a small decomposition level. After the qualitative reasoning, the algorithm continues to a quantitative reasoning if there are quantitative design requirements. Candidate mechanism “CM” will be retracted from the candidates list. Then, “CM” will insert a set of transformation expressions (translation, rotation, and scaling), and all the symbolic expressions of “CM” (if there are any) with undetermined variables are placed in a constraint store in the SICStus Prolog for constraint satisfaction.

When a constraint is inserted, constraint handling rules are fired to simplify or propagate the constraints in the constraint store. After all constraints are inserted, "CM" instantiates design parameters obtained from its instance and calls corresponding CLP optimization predicates to obtain feedbacks and asks for a decision from the designer, and then "CM" will remove the constraints inserted by itself from the constraint store. If there are solutions fitting the required motion parameters of both "inputmotion" and "outputmotion", then "Mech" is unified with "CM", and the detailed information of "Mech" is displayed through a predicate "displayprotoinfo/0". Otherwise, "CM" is ranked by the user and inserted into a ranked child list of "AM", and "AM" continues to check the next candidate child.

6. If "AM" cannot find any child that can be transformed to suit the requirements, then the design task is decomposed into:

```
AM::retractrankedsub(CM),
CM::insertconstraints,
SISO_Syn(inputmotion, CM::inputmotionattr, constraints,
     Mech1),
append(Mech1, CM, Mech).
```

The predicate "retractrankedsub/1" retracts a child "CM" with the highest rank from the ranked child list of "AM". "CM" then inserts its constraints into the constraint store. These constraints will not be removed until the backtracking returns to the point where "CM" is retracted. Now, the constraints in the constraint store will be populated as the decomposition level increases. Note that either a specific decomposition level or a specific optimization standard can be used to stop undesirable decomposition and force a backtracking. Since the current CLP is still weak at solving non-linear constraints, the algorithm will prompt the user whether or not it is necessary to insert a non-linear. Further, some heuristics are embedded in the algorithm to prevent composition explosion, such that the number of similar primitive mechanisms with interchangeable motion transfer direction is limited to a predefined number. The algorithm extends our previous algorithm with constraint solving capability.

The synthesis of SIMO mechanism can be solved by applying the SISO_Syn algorithm for each pair of output motion and input motion, and then combining these solutions. The method is not optimum but is still acceptable, since the number of outputs in most SIMO mechanisms is smaller than four. Note that there may be redundant components, since the topological branch point is limited to the root input. Therefore, another algorithm is provided to enumerate non-redundant structures for each solution.

In our approach, while the synthesis of the MIC mechanism is still derived from the synthesis of the SISO mechanism, it is much different from traditional approaches. It is strongly related to the global layout type of the mechanism structure. Indeed, few approaches have considered the conceptual design of such mechanisms. Direct forward reasoning without considering the global layout is generally impossible due to the combinatorial explosion. We prefer a prototype-based synthesis. The designer selects an existing prototype and abstracts its global layout. Each serially-connected sub-mechanisms chain in the prototype is collected as a single abstract SISO mechanism. The corresponding motion constraints and transformation constraints between the abstract SISO mechanisms are described when possible. For the synthesis of a compound mechanism with complicate spatial constraints between its abstract SISO mechanisms, there are still many difficulties. Fortunately, in many cases, the output motion of each abstract SISO mechanism and the transformation constraints between the abstract SISO mechanisms are qualitatively describable. Therefore, the synthesis of the MIC mechanism can be carried out by the parallel synthesis of those constraints. Note that it is possible to produce a single input compound (SIC) mechanism by concatenating a MIC mechanism with a corresponding SIMO mechanism.

4.2 Topological reasoning of structures

An important task in the synthesis of MIC mechanisms and SIC mechanisms is the innovative design of its structure and substructures. Such a task is generally not important in the synthesis of SISO mechanisms. This design task leads to the topological reasoning of structures.

As aforementioned, type synthesis techniques are relatively well-developed. We adopt the type synthesis algorithm presented in [8]. The algorithm employs Polya's theory for the enumeration of structure types. A modification is that the kinematic pair types of the inputs and outputs in the original prototype will not be changed; this means that a prismatic pair will not be changed to a revolute pair. Nevertheless, newly grounded members with arbitrary kinematic pair types can still be added. The drawback of the modification is that the enumeration may miss some innovative structures. On the other hand, the synthesis algorithm becomes more tractable in the entire synthesis process. The enumerations of sub-structures are carried out after the behavioural reasoning of the MIC mechanisms. Currently, we assume that only the last sub-mechanism of each abstract SISO mechanism in the MIC mechanism requires topological reasoning. Motion constraints and transformation constraints will be checked again when a new sub-structure is generated. Each successful new substructure will be instantiated as a new prototype and inserted as a cousin of the current prototype mechanism being synthesized.

The above reasoning can solve a number of mechanism synthesis problems. It has yet to be noted that the efficiency and flexibility of the above synthesis algorithms are still not desirable for large systems due to the nature of plain backtracking. We are currently conceiving corresponding dependency-directed backtracking algorithms to address the efficiency problem. Moreover, we are investigating how to integrate the above reasoning techniques for designing multiple inputs–multiple outputs (MIMO) mechanisms with coordinated motions.

5 Case study

We selected the conceptual design of a zigzag sewing machine to demonstrate the performance of the integrated knowledge representation model and its reasoning mechanism. A zigzag sewing

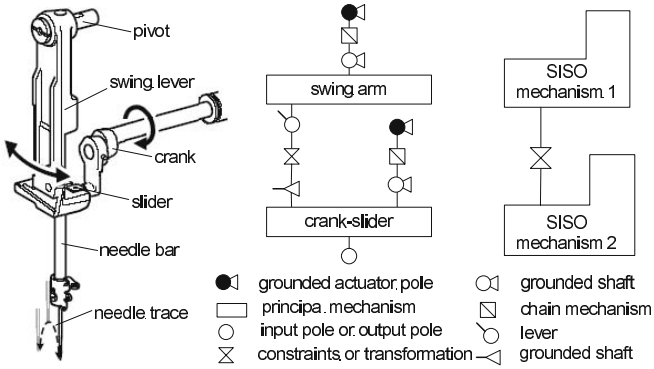


Fig. 3. A prototype of a zigzag mechanism and its global topology

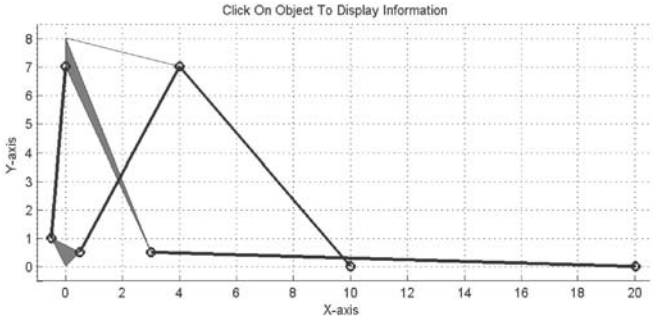
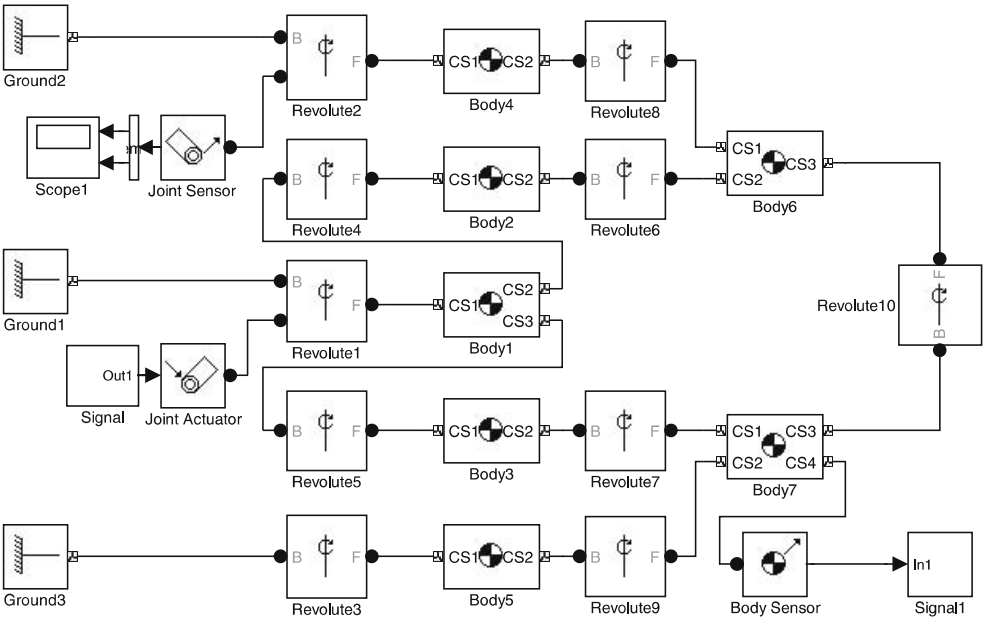


Fig. 5. Sketch of a new zigzag mechanism

machine has a zigzag mechanism whose performance is crucial to decide the sewing quality. In our early research, a number of sewing machine patents have already been analyzed and their kinematic information and topology information have been encoded into the external database. A designer can retrieve zigzag mechanism prototypes by keywords. Note that the global layouts of all the zigzag mechanisms are the same. There are two main sub-mechanisms: a positioning mechanism oscillating from side to side, and a feeding mechanism reciprocating the needle up and down. A typical zigzag prototype is illustrated in Fig. 3. There are two problems with this prototype: (a) there is a needle deflection if the swing angle of the positioning mechanism is large - this is a minor problem; and (b) the dwell time of the positioning mechanism on both sides of the needle trace is not long enough. Thus, the major design task is to redesign the positioning mechanism which satisfies the geometrical and operational constraints between itself and the feeding mechanism. Other design constraints are added, such as the fact that cams and gears are not used, since the sewing machine will run at a very high speed. The sub-mechanism of the needle reciprocating mechanism is fixed as a crank-slider. The topology information and kinematic information of the prototype and the design constraints are processed to obtain a series of prolog predicates. The console loaded the predicates to the SICStus Prolog, and after a series of interactions the prolog system returned a number of concepts including four existing prototypes of four-bar linkages, six existing prototypes of six-bar linkage, two new prototypes of six-bar linkages, and 18 new prototypes of eight-bar linkages. All the existing prototypes have design strategies for designing dwell motions, while those design strategies cannot be applied to the new prototypes. Hence, coarse optimizations are necessary. Among all these linkages, we selected a Stephenson-II-type six-bar linkage and a double butterfly-type eight-bar linkage as examples of coarse dimensional design employing hybrid genetic algorithms. The coarse optimization of the six-bar linkage fails without obtaining qualified solutions, while the coarse optimization of the eight-bar linkage obtains eight solutions with dwell motions approximating that of the existing Stephenson-III type prototypes, and can be used for future dimensional synthesis. Finally, constraints checking is successful, since the motion constraints and

Fig. 4. Detailed block diagram of a new zigzag mechanism



transformation constraints are all simple planar constraints. Figures 4 and 5 offer a detailed block diagram and the sketch of an obtained eight-bar linkage, respectively. Note that there are another two types of zigzag prototypes with different transformation constraints. The same procedures are repeated, and another 14 candidates are obtained.

6 Conclusions

An integrated knowledge representation model for the computer-aided conceptual design of mechanisms was presented in this paper. Besides modelling the functional, behavioural, and structural aspects of mechanisms, the model also considers the complex topological characteristics of mechanisms that are normally absent in general physical devices. The model has a multi-level architecture to improve its reasoning efficiency, and allows for a clean integration of a qualitative reasoning with a quantitative reasoning and simulation. Compositional behaviour reasoning and automatic topology generation have been integrated to develop a relatively complete system for the computer-aided conceptual design of mechanisms. Nevertheless, current reasoning in the functional level is still limited to prototype retrieval. Furthermore, the interface of the geometrical reasoning of the configuration space for the design of multiple-state mechanisms was considered, but not realized due to the challenges of geometrical reasoning. These problems will be investigated in future research.

Acknowledgement The research is supported by the Chinese National Science Foundation (Grant No. 50275092), and was partially sponsored by the Shanghai Industrial Sewing Machine Co. Ltd.

References

1. Zou HJ (2002) Conceptual design of mechanical systems. Mechanical Engineering Publisher, Beijing (in Chinese)
2. Johnson RC (1978) Mechanical design synthesis – creative design and optimization. Krieger, Melbourne, Florida
3. Kota S (1993) Type synthesis and creative design of mechanisms. In: Erdman AG (ed.) Modern kinematics, developments in the last forty years. Wiley, New York, pp 27–74
4. Hsu W, Woon IMY (1998) Current research in the conceptual design of mechanical products. Comput Aided Des 30(5):377–389
5. Freudenstein F, Dobrjanskyj L (1964) On the theory for the type synthesis of mechanisms. In: Proceedings of the 11th International Conference on Applied Mechanics, pp 420–428
6. Mruthyunjaya TS (2003) Kinematic structure of mechanism revisited (review). Mech Mach Theory 38:279–320
7. Tsai LW (2001) Mechanism design — enumeration of kinematic structures according to function. CRC Press, Boca Raton, Florida
8. Yan HS (1998) Creative design of mechanical devices. Springer, Singapore
9. Stahovich TF (2001) Artificial intelligence in design. In: Antonsson EK, Cagan J (eds) Formal engineering design synthesis. Cambridge University Press, Cambridge, pp 228–269
10. Joskowicz L, Sacks E (1991) Computational kinematics, Artif Intell, 51(1–3):381–416
11. Artobolevsky I (1979) Mechanisms in modern engineering design, volume 1–4. MIR Publishers, Moscow (English translation)
12. Joskowicz L, Neville D (1996) A representation language for mechanical behaviour. Artif Intell Eng 10(2):109–116
13. Subramanian D, Wang CS (1995) Kinematic synthesis with configuration space. Res Eng Des 7:193–213
14. Kannapan S, Marshek K (1990) An algebraic and predicate logic approach to representation and reasoning in machine design. Mech Mach Theory 25:335–353
15. Li CL, Tan ST, Chan KW (1996) A Qualitative and heuristic approach to the conceptual design of mechanisms. Eng Appl Artif Intell 9(1):17–31
16. Chiou SJ, Kota S (1999) Automated conceptual design of mechanisms. Mech Mach Theory 34(3):467–495
17. Ye ZG, Zou HJ, Wang SZ (2004) The establishment and reasoning of knowledge base system for mechanism kinematic schemes. Int J Adv Manuf Technol 23:295–300
18. Forbus KD (1980) Spatial and qualitative aspects of reasoning about motion. In: Proceedings of the AAAI-80, AAAI Press, Menlo Park, CA
19. Faltings B (1990) Qualitative kinematics in mechanisms. Artif Intell 44(1–2):89–119
20. Faltings B, Sun K (1996) FAMING: supporting innovative mechanism shape design. Comput Aided Des 28(3):207–216
21. Stahovich TF, Davis R, Shrobe H (1998) Generating multiple new designs from a sketch. Artif Intell 104:211–264
22. Li CL, Chan KW, Tan ST (1999) A configuration space approach to the automatic design of multiple-state mechanical devices. Comput Aided Des 31:621–653
23. Roy U, Pramanik N, Sudarsan, R, Sriram RD, Lyons KW (2001) Function-to-form mapping: model, representation and applications in design synthesis. Comput Aided Des 33:699–719
24. Bracewell RH, Sharpe JEE (1996) Functional descriptions used in computer support for qualitative scheme generation – "Schemebuilder". Artif Intell Des Anal Manuf 10(4):333–346
25. Molian S (1969) Storage and retrieval of descriptions of mechanisms and mechanical devices according to kinematic type. J Mech 4: 311–323
26. Hoeltzel DA, Chieng WH (1990) Pattern matching synthesis as an automated approach to mechanism design. J Mech Des 112:190–199
27. Kota S (1992) Automatic selection of mechanism designs from a three-dimensional design map. J Mech Des 114:359–367
28. Swedish Institute of Computer Science (2003) SICStus Prolog 3.11.0 User's Manual. Swedish Institute of Computer Science, Kista, Sweden
29. Tian YL, Zou HJ, Guo WZ (2003) Study on modelling and simulation methods for constitution objects of mechatronic products based on Matlab-SimMechanics. Mach Des Res 19(5):10–14 (in Chinese)