Knowledge-based Inference Methods for Modeling Technical Systems

Gerd Kamp and Bernd Neumann
University of Hamburg, Vogt-Kölln-Str.30, D-22527 Hamburg
Email:{kamp,neumann}@informatik.uni-hamburg.de

Abstract

Description Logics with concrete domains present an approach to realize a general engineering workbench. They provide a representation language that enables us to describe in a uniform way devices, assemblies and components along with their structure, constraints on attributes and physical laws as well as models of their correct and faulty behavior. Furthermore, sound and complete algorithms can be given for a set of basic inferences. These basic inferences render it possible to simulate the behavior of the devices and provide the basic building blocks for consistency-based diagnosis. In addition they enable us to devise procedures for finding errors, omissions and inconsistencies in model libraries.

1. Introduction

During the design and engineering phase of a new technical device it is necessary to develop solutions for several tasks, such as configuration from components and assemblies, determination of parameter values, simulation of behavior, intelligent selection from parts catalogs and diagnosis of prototypes. Traditionally these problems are tackled with a number of specific tools. These tools most often be stand-alone systems and are not designed to communicate with each other. In other words, every tool realizes its own language to describe the device, thereby preventing results obtained by one tool to be employed in other tools.

In this paper we present an approach for an integrated engineering workbench based on description logics extended with concrete domains. The knowledge representation language that is supplied by this approach enables us to describe important model features in a uniform language, e.g.: component types, component structures, physical laws and models of correct and faulty behavior. A number of basic inference services can be defined, and sound and complete algorithms for these inferences can be given for the description language used. First results show that by making use of these basic inference services it is not only possible to simulate the behavior and support the diagnosis of the system, but also to detect errors, inconsistencies and lacunae in model libraries.

The remainder of this paper is organized as follows: First we give a brief introduction to description logics and the following enhancements needed in technical domains: concrete domains, model generation and calculation of admissible parameter ranges. We then focus on how different engineering tasks can be tackled with this approach. In Section 2 we show how the basic concepts needed to describe gearwheel mechanisms can be represented within the description language. Section 3 shows how simulation behavior can be accomplished via model generation and calculation of admissible parameter values. Section 4 presents an approach to diagnosis based on object classification. Section 5 shows how the inference service of concept classification is useful for building and maintaining large model libraries. A summary and an outlook conclude the paper.

1.1. Description Logics

Description logics (DL) have a long tradition in organizing information with a powerful object-oriented representation scheme and clearly defined semantics. Description logics systems mainly consist of two parts: a Terminological Box (TBox) and an Assertional Box (ABox)[1]:

1.1.1. TBox At the core of description logics lies the notion of *concept terms*. Starting with primitive concept and role terms, new concept terms can be constructed from others by a set of concept forming operators. There are mainly the following categories of such operators [6]:

1. *Boolean operators* (and *C D* ...), (or *C D* ...), (not *C*), allowing for the combination of concepts without a reference to their internal structure.
2. *Role forming operators* that allow new roles to be defined, e.g. composition of roles (compose *r s*).
3. *Operators on role fillers* that operate on the internal structure of the concept terms, e.g. provide quantification over role fillers (some *r C*).

Terminological axioms of the form (define-concept CN *C*) associate a concept name CN with a concept term *C* and are used to define the relevant concepts of an application.

[1] We cannot elaborate on the basics of description logics and refer the reader to [1] for a more detailed introduction.

1060-3425/97 $10.00 © 1997 IEEE

Terminological axioms are roughly comparable to the class definitions of an object-oriented representation language[2]. Finally a *TBox* $\mathbf{T}$ is a finite set of terminological axioms.

1.1.2. ABox Concrete objects are realized as instances of concepts. New instances o can be introduced into the ABox via (define-distinct-individual o), and assertions concerning the membership of an instance o to a concept C, or about existing relations r between two objects o and p can be made through (state (instance o C)) resp. (state (related o p r)). The set of assertions finally constitutes the ABox $\mathbf{A}$.

1.2. Basic Inference Services

What sets description logics apart from other knowledge representation approaches, is that one is able to formally define a model-theoretic semantics by means of an interpretation function I. This formal semantics allows a formal definition of a number of powerful inferences. In our context the following inference services are of particular interest:

Classification Classification is the most prominent TBox inference service. Classification calculates the concept subsumption hierarchy, i.e. the subconcept-superconcept relations between pairs of concepts. Technically a concept C subsumes another concept D if each model for C is also a model for D (i.e. $I(C) \supseteq I(D)$).

Object Classification Object Classification is an ABox inference service that, given an object o of the ABox, determines the set of most specific concepts in the concept hierarchy $\{C\ D \ldots\}$ to which this object is a member.

Retrieval The retrieval problem is dual to the Object Classification problem. Here the set of ABox objects (instances) $\{o\ p \ldots\}$ are returned that are members of a given concept C.

All these inference services can be reduced to the *consistency* problem of an ABox. An ABox is consistent if it has a model, i.e. the set of interpretations $I(\mathbf{A})$ is not empty (see e.g. [2] for details). Further it can be shown that there exist sound and complete algorithms for certain description languages, especially for the language *ALCF* which we have chosen for our system (for a description of *ALCF* see [6]).

1.3. Concrete Domains

The previous section more or less gave the general framework for a standard description logics system. But in order to use description logics in technical domains more expressive representation languages are needed. In addition to the *abstract domain* of definable concepts presented above several *concrete domains*, such as numbers, strings and symbols must be added to the language and the inference services of the description logic[3]. These are needed in order to describe parameter values and constraints between different parameters.

For example, in order to represent, simulate and diagnose the simple bike drive train in Fig.1 at least the following knowledge must be representable within such a system:

1. The different *types of components*, e.g. wheels, gearwheels, chains, along with their *attributes* like force F, radius r and torque M.[4]
2. The *structure* of assemblies, e.g the kinematic structure of the mechanisms using kinematic pairs.
3. *Physical laws* like $M = F{\cdot}r$ and *constraints* imposed on the attributes like $F>=0$ and $r>0$.
4. The *normal and faulty behaviors* (often called models in consistency-based diagnosis) of components and assemblies, e.g. the propagation of torques from one wheel to another ($M1=M2$) with a correct or faulty chain drive.

Therefore, in order to describe physical laws and models of behavior we need at least a concrete domain for which one is able to reason with linear systems of inequalities between multivariate polynomials. But current terminological systems such as LOOM, CLASSIC, KRIS and TAXON realize only a concrete domain [9] where it is possible to express comparisons of parameters with constant values.

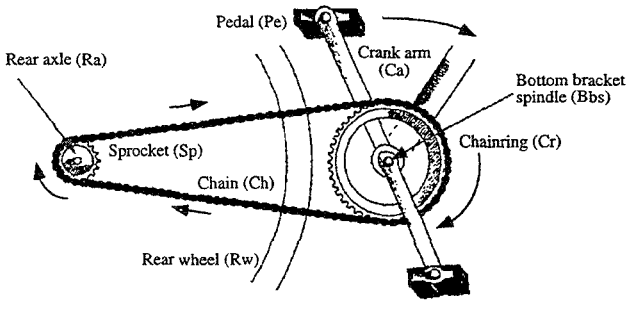

Figure 1: **A simple bike drive train**

A scheme for the integration of *admissible* concrete domains was developed by Baader and Hanschke [2, 6]. An

[2]The class hierarchy of object oriented systems could be translated into conjunctive concept terms with the slot definitions resulting in appropriate role definitions.

[3]This requirement arises in other domains, too. But in technical domains it is vital.

[4]In this paper we model these attributes using scalar numeric values resulting in quantitative models of behavior. Alternatively a qualitative model using symbolic values as it is used in qualitative physics is possible.

admissible concrete domain is mainly a base data type together with a set of so-called concrete predicates (n-ary relations over the base data type). In order to be admissible such a structure has to fulfill a number of constraints. The most important of these constraints requires that the satisfiability of a finite conjunction of concrete predicates must be decidable. Baader and Hanschke then show that if a concrete domain is admissible the resulting concept language *ALCF(D)* is still decidable. Hence, there exist sound and complete algorithms for the above inference services in *ALCF(D)*. To our knowledge, TAXON is the only description logic system that uses this approach to realize concrete domains. But as already mentioned, so far the concrete domains are too inexpressive.

Fortunately it can be shown [9] that systems of inequalities are an admissible concrete domain. Therefore it is possible to build a description logic system that is able to represent and handle the above mentioned component types, the laws of physics and the models of behavior. Thus we developed CTL, a system where admissible concrete domains are realized through a well-defined interface to external algorithms [9]. In particular, we are currently able to handle systems of arbitrary linear polynomials with the help of an CLP(R)-system.

In order to express concrete predicates we extended the description language with the following constructs: (define-constraint *PN* (*x1* ... *xN*) expr) defines a new concrete predicate (normally an (in)equality) between a number of variables. Further (constrain *R1* ... *RN P*) is an additional concept term operator for associating a number of parameters *R1* ... *RN* with the variables of a defined concrete predicate *P*. Alternatively, it is possible to write (constrain *R1* ... *RN* ((*x1* ... *xN*) expr)) if one does not want to introduce a predicate name but rather use anonymous concrete predicates.

1.4. Model Generation

Description logic systems with sound and complete inference algorithms are based on a tableaux calculus. This means that in order to determine if an ABox is consistent, the consistency test explicitly generates a consistent model. That model obeys and enforces the constraints imposed on the ABox objects by the TBox concepts. Not only are the restrictions enforced on the ABox instances introduced by the user, but additional instances may be generated by operators on role fillers, especially the operator some. The latter fact comes in handy, e.g. if one wants to check if there are any instances that were automatically generated and not introduced by the user. It is also the base for configuration systems and useful during the construction phase of a device model in simulation and diagnosis.

Description logic systems normally display only the result of the consistency test and the constructed model is more or less thrown away. But at least in technical domains we are not only interested in the existence of a solution, we want to see the solution, e.g. the set of instances and the relations between them. Therefore, the calculated model is the main object of interest, not the result of the consistency test. Thus we extended CTL with a method that enables us to access the model which is generated during the consistency test.

1.5. Calculating Admissible Parameter Ranges

If a description logic is enhanced with concrete domains, it must provide a means to check the consistency of the concrete domain. As we have seen this is mainly the task of checking whether a finite conjunction of predicates is satisfiable. In our case this means checking if a system of inequalities between polynomials has a solution. But in analogy to model generation we are not only interested in the existence of a solution, but in how this solution looks like.

Both tasks can be accomplished by using quantifier elimination techniques from computer algebra (in the field of CLP(R) this technique is called variable elimination or projection). Quantifier elimination [17] is a method that transforms an arbitrary first-order formula of the theory of the elementary algebra over the reals (more or less "real arithmetic") into an equivalent formula without (or with fewer) quantifiers. Since the solvability of a system of inequalities can be expressed as an existential sentence of elementary algebra (for details see [9]), checking for the existence of a solution of such a system is simply done by eliminating all quantifiers from that sentence.

Besides checking the validity of a sentence, quantifier elimination can also be used for the calculation of admissible parameter ranges. This is realized by eliminating all parameters but the one whose restrictions should be calculated. Quantifier elimination transforms the sentence into an equivalent one containing only one quantified variable. The resulting sentence describes the restrictions on the range of the respective parameters admissible values. Since this procedure is independent of the actual parameter, it can be used to determine the admissible ranges of all parameters contained in the model that is generated during the consistency test.

Over the years quantifier elimination techniques have been vastly improved. On one side more efficient general algorithms than Tarskis have been devised (e.g. Cylindrical Algebraic Decomposition (CAD) [3]) and further improved (e.g. PCAD [7]). But these are still too inefficient to be actually useful and no implementation is publicly available. On the other side specialized algorithms for subsets of the theory of elementary algebra have been found and implemented. Most important, more or less all CLP(R)-systems implement some kind of the Fourier-Motzkin algorithm that realizes quantifier elimination for linear systems of inequalities (see e.g. [11]). Since quite a number of

CLP(R)-systems are freely available and are sufficient for our representation needs, we have chosen a CLP(R)-system as the decision procedure over systems of linear inequalities. CLP(R)-systems can also be used for nonlinear systems, but in this case complete inferences can not be guaranteed. Since the only non-linear expression in the above examples is the law of torque, this restriction does not really impose a problem on our example. At the moment we are implementing an interface to a quantifier elimination procedure of a computer algebra system. This elimination procedure allows us to handle arbitrary quadratic sentences of the theory of the elementary algebra over the reals. With this system we can guarantee sound and complete inferences for our example[5].

In the following we will use a simple example from mechanics to illustrate how description logics based on CTL can be used for the representation, the simulation and the diagnosis of simple linear systems.

2. Representation

The first task is to describe the different kinds of knowledge to be captured by the provided description language. First, we must be able to describe the different component types of a mechanism as well as the kinematic structure. In kinematics this is normally done with links and kinematic pairs [16, 8]. In the following we show how these concepts can be described in the description language of CTL, which is based on the proposed KRSS standard [15] for description logic languages.

2.1. Links

In order to describe the different component types of the drive train of Fig.1, it is sufficient to define rotational links and tension links as specializations of general links. The terminology in Fig.2 defines links as something that carries a force (link.force). Rotational links (rotational-link) are links that in addition have attributes for a radius (rot.radius) and a torque (rot.torque). link.force and rot.torque are not negative, rot.radius is strictly positive, and the torque is the product of radius and applied force. Finally we define a number of additional links such as wheel, chain etc. without detailing them further.

2.2. Kinematic Pairs

In order to describe the behavioral structure of a mechanism, kinematic pairs are used. A kinematic-pair describes the connection between two links pair.link1 and pair.link2. Depending on the relative motion of the links and the type of connection different types of pairs can be identified. The

[5]We already used this system to calculate the parameter restrictions in the following examples

```
(define-primitive-attribute link.force Top)
(define-primitive-concept link
  (and Top (some link.force (minimum 0))))
(define-primitive-attribute rot.radius Top)
(define-primitive-attribute rot.torque Top)
(define-constraint x>0 (?x) (> ?x 0))
(define-constraint x>=0 (?x) (>= ?x 0))

(define-constraint x*y=z (?x ?y ?z)
   (= (* ?x ?y) ?z))
(define-primitive-concept rotational-link
  (and link
  (constrain rot.radius x>0)
    (constrain rot.torque x>=0)
    (constrain rot.radius link.force rot.torque
        x*y=z)))

(define-primitive-concept tension-link link)
(define-primitive-concept crank rotational-link)
(define-primitive-concept wheel rotational-link)
(define-primitive-concept gearwheel
                          rotational-link)
(define-primitive-concept spindle rotational-link)
(define-primitive-concept chain tension-link)
```

Figure 2: **Links**

terminology shown in Fig.3 is restricted to the description of pairs of two rotational-links (rot-pair) and pairs of a rotational-link and a tension-link (rot-tension-pair). Note that it is possible to describe rot-tension-pair via an or construct, something which is not possible within object-oriented representation systems.

```
(define-primitive-attribute pair.link1 top)
(define-primitive-attribute pair.link2 top)
(define-primitive-concept kinematic-pair
    (and (all pair.link1 link)
         (all pair.link2 link)))

(define-concept rot-pair
    (and kinematic-pair
      (all pair.link1 rotational-link)
      (all pair.link2 rotational-link)))

(define-concept rot-tension-pair
  (and kinematic-pair
    (or (and (all pair.link1 rotational-link)
             (all pair.link2 tension-link))
        (and (all pair.link2 rotational-link)
             (all pair.link1 tension-link)))))
```

Figure 3: **Kinematic pairs**

2.3. Models of Behavior

In addition to the component types and the structure of the device, descriptions of the correct and of different faulty behaviors are needed for the consistency-based diagnosis as well as the simulation of the device. The terminology in

Fig.4 describes the correct behavior of rotational-pairs and rot-tension-pairs. The torque is propagated in ok-rot-pair, the force is propagated in ok-rot-tension-pair.

```
(define-constraint x=y (?x ?y) (= ?x ?y))
(define-concept ok-rot-pair
  (and rot-pair
    (constrain (compose pair.link1 rot.torque)
               (compose pair.link2 rot.torque)
               x=y)))

(define-concept ok-rot-tension-pair
  (and rot-tension-pair
    (constrain (compose pair.link1 link.force)
               (compose pair.link2 link.force)
               x=y)))
```

Figure 4: **The correct behavior of the kinematic-pairs**

The terminology in Fig.5 shows some exemplary faulty behaviors of a rotational pair. A pair is slipping (slipping-rot-pair) if both torques are strictly positive and different. A pair is broken (broken-rot-pair) if one torque is strictly positive, the other zero.

```
(define-concept slipping-rot-pair
  (and rot-pair
    (constrain (compose pair.link1 rot.torque)
      x>0)
    (constrain (compose pair.link2 rot.torque)
      x>0)
    (or (constrain (compose pair.link1 rot.torque)
                   (compose pair.link2 rot.torque)
          x>y)
        (constrain (compose pair.link2 rot.torque)
                   (compose pair.link1 rot.torque)
          x>y))))

(define-constraint x=0 (?x) (Number) (= ?x 0))
(define-concept broken-rot-pair
  (and rot-pair
    (or
     (and
      (constrain (compose pair.link2 rot.torque)
        x>0)
      (constrain (compose pair.link1 rot.torque)
        x=0))
     (and
      (constrain (compose pair.link1 rot.torque)
        x>0)
      (constrain (compose pair.link2 rot.torque)
        x=0)))))
```

Figure 5: **Faulty behaviors of a rotational pair (I)**

A second developer might distinguish between strong and weak slipping pairs (strong-slipping-rot-pair resp. weak-slipping-rot-pair) as it is depicted in Fig.6. Note that in our language it is possible to describe the weak slipping pair as the negation of a strong one. This allows for a simple description of a weak slipping pair, and reduces the sources of possible faults. It further eases the modification of the knowledge base, e.g. a change of the limiting value between strong and weak slipping pairs is local to one definition and not spread across two definitions.

```
(define-constraint x<.3y (?x  ?y)
  (< ?x (* 3/10 ?y)))
(define-concept  strong-slipping-rot-pair
  (and rot-pair
   (constrain (compose pair.link1  rot.torque)
     x>0)
   (constrain (compose pair.link2 rot.torque)
     x>0)
   (or
    (constrain (compose pair.link1 rot.torque)
               (compose pair.link2 rot.torque)
      x<.3y)
    (constrain (compose pair.link2  rot.torque)
               (compose pair.link1 rot.torque)
      x<.3y))))

(define-concept weak-slipping-rot-pair
  (and slipping-rot-pair
    (not  strong-slipping-rot-pair)))
```

Figure 6: **Faulty behaviors of a rotational-pair (II)**

Fig.7 depicts the concept hierarchy after concept classification of the terminologies shown in Fig.2 through Fig.6.

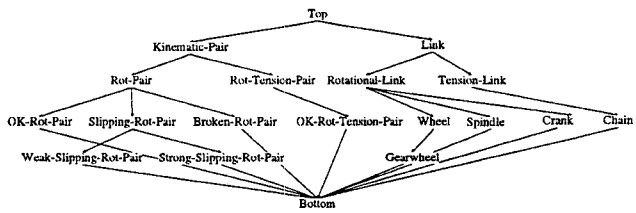

Figure 7: **The classified TBox**

3. Simulation of Behavior

In this section we illustrate how model generation based on the consistency test together with parameter range calculation can be used for the simulation.

As we have seen, one part of the consistency test is to check whether the constraints which are implied by the assertions in the ABox are satisfiable. This holds true especially with respect to constraints specified in the definitions of behavior models and links. If for example we assert that link-1 is a rotational link with (state (instance link1 rotational-link)), the law of torque must hold for link-1 in a consistent ABox. Additionally, it must be true that the force and the torque are positive and that the radius is strictly positive.

In order to check all these constraints a constraint net in form of a system of inequalities is constructed in the course

of the consistency test. All constraints are automatically entered into this constraint net and its satisfiability is automatically checked. If one additionally gives the values of some parameters (e.g. via (state (related link1 10 link.force)) these values are also automatically entered into the constraint net. Furthermore, every assertion about parameters constrains the range of admissible values of at least this parameter, but most often also a number of additional parameters (e.g. via the law of torque).

Using this approach, it is unnecessary to define a fixed set of input parameters for the simulation. The consistency test and hence the model generation and calculation of parameter ranges works whichever set of assertions is given. Since the consistency test can be initiated after every ABox assertion, i.e. after every (state ...), this approach renders it possible to interactively and incrementally determine the model and the parameter ranges. Moreover it is possible to freely intermix assertions about parameter values (e.g. (state (related link1 10 link.force))) and behavior models (e.g. (state (instance pair1 ok-rot-pair)), giving the developer a maximum amount of freedom. We illustrate this procedure using the bike drive train from Figure 1 as an example.

3.1. Describing the device

Before a device can be simulated, its structure must be described. To do so, firstly the instances are created and appropriate primitive concepts are assigned to the freshly created instances:

```
(define-distinct-individual crankarm-1)
(define-distinct-individual chainring-1)
(define-distinct-individual
   bottom-bracket-spindle-1)
(define-distinct-individual chain-1)
(define-distinct-individual sprocket-1)
(define-distinct-individual rear-axle-1)
(define-distinct-individual rear-wheel-1)
(state
 (and (instance crankarm-1 crank)
      (instance chainring-1 gearwheel)
      (instance bottom-bracket-spindle-1
        spindle)
      (instance chain-1 chain)
      (instance sprocket-1 gearwheel)
      (instance rear-axle-1 spindle)
      (instance rear-wheel-1
        rotational-link)))

(define-distinct-individual pair1)
...
(define-distinct-individual pair6)
(state
 (and (instance pair1 kinematic-pair)
...
      (instance pair6 kinematic-pair)))
```

Secondly, the links are related to the respective kinematic pairs and the kinematic chain is set up:

```
(state
 (and
  (related pair1 crankarm-1 pair.link1)
  (related pair1 bottom-bracket-spindle-1
           pair.link2)
  (related pair2 bottom-bracket-spindle-1
           pair.link1)
...
  (related pair5 rear-axle-1 pair.link2)
  (related pair6 rear-axle-1 pair.link1)
  (related pair6 rear-wheel-1 pair.link2)))
```

This concludes the description of the structure of the drive train. Since neither parameter values nor behavior models are given at this point, the generated model only reflects the constraints imposed on the parameter values by the concept definitions of their respective concepts (see Figure 8).

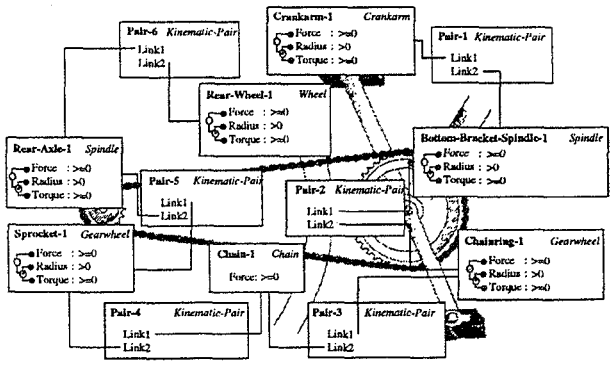

Figure 8: **Simulation: Describing a device**

3.2. Simulation with parameter values

The simplest form of simulation uses exact parameter values for some parameters in order to determine the admissible parameter ranges for the other parameters contained in the calculated model. If there are no disjunctions, or inequalities are contained in the models of behavior, the ranges for the dependent parameters are restricted to a single exact value. Simulation proceeds as follows:

1. Determine of some parameter values in order to start the simulation.

   In our example we set the radii of the crankarm crankarm-1, the chain ring chainring-1, the sprocket sprocket-1 and the rear wheel rear-wheel-1, as well as the value of the force applied to the crank arm:

```
(state (and
  (related crankarm-1 .175 rot.radius)
  (related chainring-1 .1 rot.radius)
  (related sprocket-1 .05 rot.radius)
  (related rear-wheel-1 0.6858 rot.radius)))
```

2. Determine of the behavior models for the components.

   In our example all kinematic pairs (pair1 ... pair6) expose their normal behavior:

```
(state
 (and (instance pair1 ok-rot-pair)
      (instance pair2 ok-rot-pair)
      (instance pair3 ok-rot-tension-pair)
      (instance pair4 ok-rot-tension-pair)
      (instance pair5 ok-rot-pair)
      (instance pair6 ok-rot-pair)))
```

   These assertions add new constraints to the constraint net (see Figure 9) but executing the consistency test does not reveal any new information since giving only some radii does not trigger the restriction of any parameter within the constraint net.

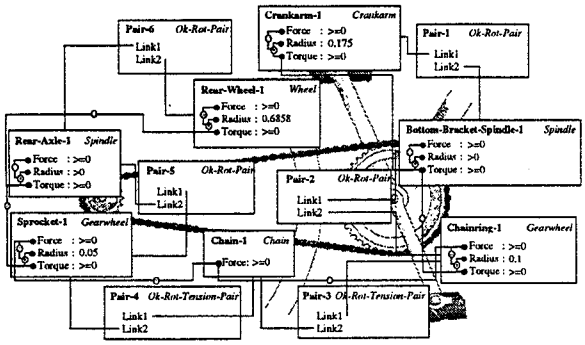

Figure 9: **Simulation with parameter values: Initial parameters and behaviors**

3. Determine some additional parameter values.

   With additional parameter values enough information is available to trigger propagation through the constraint net. In our example we choose to assert a value for the force that is applied to the crankarm by stating[6]:

```
(state (related crankarm-1 200 link.force))
```

   Now the values for nearly all missing parameter values can be calculated (see Figure 10). W.r.t. the bottom-bracket-spindle-1 and the rear-axle-1 the value 0 could be excluded from the admissible ranges of the respective forces (since the force applied to the crankarm is strictly positive). It is not surprising that no further restriction is possible, given the fact that no radius is given for these links. Nevertheless the propagation of the respective torques to their adjoining links is possible because the law of torque is applicable to intact rotation pairs.

[6] 200N is a reasonable value for an average cyclist.

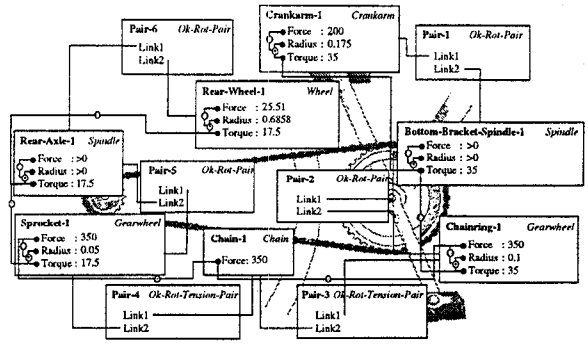

Figure 10: **Simulation with parameter values: Final Model**

3.3. Simulation with complex parameter restrictions

We now consider simulation using complex parameter restrictions instead of simple parameter values. The quantifier elimination method allows us to use arbitrary concrete predicates as parameter restrictions, i.e. linear (resp. quadratic) sentences over the elementary algebra of the reals.

This kind of simulation is especially interesting for performing What-If analysis during the design phase of a technical device. But it allows us also to describe more complex devices. The following example combines both aspects. Consider a drive train that consists not only of a single chainring and a single sprocket but which has two chainrings and a freewheel block with sprockets in 3 different sizes. This can be described using our drive train set-up from above and the following parameter restrictions:

```
(state (and
  (related crankarm-1 .175 rot.radius)
  (related rear-wheel-1 0.6858 rot.radius)
  (instance chainring-1
    (constrain rot.radius
      ((?x) (or (= ?x 0.1)(= ?x 0.125)))))
  (instance sprocket-1
    (constrain rot.radius
      ((?x) (or (= ?x 0.05)(= ?x 0.0625)
                (= ?x 0.075)))))))
```

Additionally we restrict the force applied to the crankarm to a range typical for a moderately trained cyclist:

```
(state (instance crankarm-1
    (constrain link.force
      ((?x) (and (>= ?x 200)(<= ?x 300))))))
```

For these restrictions the model shown in Fig.11 is generated. Actually the quadratic elimination procedure delivers much more information since the different chainring freewheel combinations are handled separately. For example the formula actually computed for the force at the rear wheel (frw) is the following:

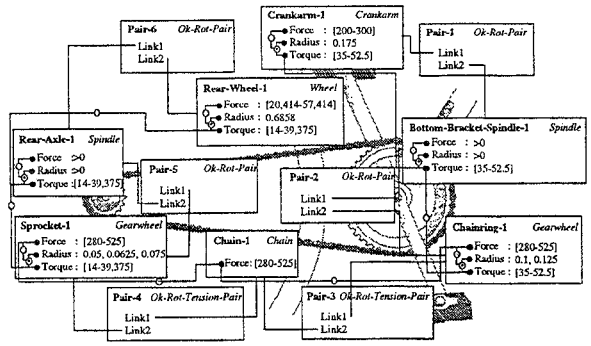

Figure 11: **Simulation with complex parameter restrictions**

```
(or (= (* 429 frw) 70000)(= (* 3429 frw) 87500)
    (= (* 3429 frw) 109375)(= (* 2286 frw) 109375)
    (= (* 1143 frw) 35000)(= (* 1143 frw) 43750)
    (= (* 381 frw) 17500)(= (* 381 frw) 21875)
    (and (> (* 1143 frw) 43750)
         (< (* 381 frw) 21875))
    (and (> (* 3429 frw) 109375)
         (< (* 2286 frw) 109375))
    (and (> (* 3429 frw) 70000)
         (< (* 1143 frw) 35000))
    (and (> (* 1143 frw) 35000)
         (< (* 381 frw) 17500))
    (and (> (* 3429 frw) 87500)
         (< (* 1143 frw) 43750)))
```

One can instantly recognize two facts from this formula. First, there are only 5 intervals instead of the expected $2 \cdot 3 = 6$. An inspection of the given radii reveals that $0.05 \cdot 0.125 = 0.625 \cdot 0.1$, and hence there are only 5 different gears. The second observation is that only 8 interval endpoints are returned, indicating that there are two occurrences of intervals that actually meet.

4. Diagnosis

As already mentioned in Section 1.2. all inferences of a description logic, such as concept- or object classification, instantiation and retrieval can be reduced to the consistency test. Object classification computes the set of concepts to which the object belongs for sure, whereas weak object classification calculates the set of concepts to which it is still possible to classify this object. These two inferences, together with a set of behavior models – models of normal and faulty behavior – can be used to realize some kind of consistency-based[7] diagnosis.

The first step is similar to the simulation approach presented in the previous section: Describing the device to be diagnosed. But rather than asserting concrete models of behavior together with a small set of input parameters (like in simulation), one starts by asserting very general models of behavior and incrementally adds values of observed parameters. After each assertion of new parameter values object classification is called. This inference service not only computes the set of most special models of behavior, but the embedded consistency test also calculates – as we have seen in the previous section – the restrictions for the various parameters. Weak object classification computes the set of behavior models to which it is still possible to classify the object. In our case this are the models of behavior that are still possible but can only be confirmed if new information is (e.g. new parameter values) available. Therefore weak classification, together with computed parameter restrictions gives hints which parameter values should be determined in the next step. Also note that weak object classification is not identical to the set of subconcepts of the concepts returned by the strong object classification. Some of these may be excluded due to the fact that they are not consistent with the calculated parameter restrictions.

Hence strong and weak object classification compute an upper and a lower boundary within the concept graph, denoting the possible models of behavior. Each new assertion of a parameter value moves these boundaries until the user has gathered enough information for a decision or both boundaries meet and the diagnosis is firm. We will illustrate this procedure by using a rotational-pair as an example:

First we describe the structure of the device (similar to simulation):

```
(define-distinct-individual rot-pair1)
(define-distinct-individual link1.1)
(define-distinct-individual link1.2)
(state (and (instance rot-pair1 kinematic-pair)
            (instance link1.2 rotational-link)
            (instance link1.1 rotational-link)
            (related rot-pair1 link1.1
                     pair.link1)
            (related rot-pair1 link1.2
                     pair.link2)))
```

Next, we assert general model of behavior:

```
(state (instance rot-pair1 kinematic-pair))
```

Finally, we assert the observed parameter values and call the object classification inference. Since different sets of parameter values should lead to different diagnosis, we illustrate the effect of this step by using three different parameter sets.

1. Giving two identical torques as parameter values leads to the classification as an ok-rot-pair, as it should be expected:

```
(state (and (related link1.1 10
                     rot.torque)
            (related link1.2 10
                     rot.torque)))
```

[7] a kind of model-based diagnosis.

2. The diagnostic process becomes clearer if one uses another set of parameter values, as shown below. The first assertion, together with the law of torque and the constraint that a radius is strictly positive enforces that the torque of link1.1 is strictly positive. Therefore the second assertion, giving a zero value for the torque of link1.2 suffices to classify rot-pair1 as a broken-rot-pair. Note that only 2 of the 6 parameters are needed for correct classification.

```
(state (related link1.2 0 rot.torque))
(state (related link1.1 8 link.force))
```

3. The last example illustrates how the rot-pair1 can be classified as a weak-slipping-rot-pair. This is possible due to the calculation of the torque of link1.2 via the law of torque and the calculation of the ratio of the two torques. This example also illustrates the role of weak object classification. After the second assertion broken-rot-pair can be excluded from the list of possible behavior models since – using the same argumentation as above – the torque of link1.2 is strictly positive. Hence both torques of rot-pair1 are strictly positive and it is impossible that it is a broken-rot-pair.

```
(state (related link1.1 20 rot.torque))
(state (related link1.2 8 link.force))
(state (related link1.2 3 rot.radius))
```

5. Model Libraries

In this section we will briefly describe how concept classification can be used for the organization and maintenance of model libraries. The following observations are important w.r.t this aspect:

1. All concept definitions are different from bottom. Therefore all definitions are satisfiable. That guarantees that no model of behavior is mistakenly defined in a way such that exists no parameter combination that may lead to this behavior (e.g. through a parameter restriction like (and (constrain force x>0) (constrain force x<0)).

2. All concept definitions are distinct from each other. This means that there are no two models of behavior that are equivalent, something which could easily happen when two model libraries are merged. If for example a third knowledge engineer has modeled a slipping rot pair as

```
(define-concept another-slipping-rot-pair
  (and rot-pair
    (constrain (compose pair.link1 rot.torque)
      x>0)
    (constrain (compose pair.link2 rot.torque)
      x>0)
    (constrain (compose pair.link1 rot.torque)
                  (compose pair.link2 rot.torque)
          x<>y)))
```

the system would detect that both definitions are equivalent.

3. The strong and weak slipping pairs in Fig.6 are modeled as specializations of rot-pair, but not of slipping-rot-pair. Situations like this may occur easily if different people develop models of behavior simultaneously, or when model libraries are complex. The classification service detects the missing subsumption relation between slipping-rot-pair and strong-slipping-rot-pair. In other situations it may be the case that a computed subsumption relation is not missing but fortuitous in a sense that it is caused by some error or laxness in the description of the models of behavior.

Errors like the ones describe above are very likely to occur in large and complex model libraries. Therefore the detection of these errors is crucial for the development of such libraries. Since all inferences are sound and complete[8] in CTL, we can guarantee that all missing and fortuitous subsumption relations in the model library are detected.

6. Summary and Outlook

Description Logics with concrete domains present an approach to realize a general engineering workbench. They provide a representation language that enables us to describe in a uniform way the devices, their assemblies and components along with their structure, constraints on their attributes and physical laws as well as models of their correct and faulty behavior. Furthermore sound and complete algorithms can be given for a set of basic inferences.

These basic inferences render it possible to simulate the behavior of the devices and provide the basic building blocks for consistency-based diagnosis. In addition they enable us to devise procedures for finding errors, omissions and inconsistencies in model libraries.

The approaches that are most similar to ours are systems that were developed within the Knowledge Sharing Effort, e.g. SHADE [13, 14, 12] and the systems derived from it like PACT [4] and PARMAN [10]. In contrast to our more basic research oriented approach, these projects investigate to which extent it is possible to define a common knowledge representation for a set of existing systems. The exchange of information is accomplished through a translation approach

[8] at least for linear systems of inequalities in our current implementation

between a common interlingua and the representation language of the target system. In practice that proved to be difficult, since different systems employ a different semantics for the same terms.

The results presented in this paper are first steps towards an integrated knowledge-based engineering workbench. Actual work focuses on realizing quantifier elimination over quadratic sentences of the elementary algebra by implementing an interface to an computer algebra system [18, 5]. Further work concentrates on concrete domains over other base types, e.g. using CLP(FD) systems for describing qualitative models. This would allow us to directly compare our approach to methods developed in qualitative physics.

Parallel to these more theoretical questions we are investigating how the basic mechanisms can be further enhanced in order to obtain systems that can actually be used by engineers. Finally we explore the possibility to integrate other modules of our envisioned workbench like configuration and intelligent retrieval from parts catalogs into the framework presented in this paper.

References

[1] F. Baader, H. Bürckert, B. Hollunder, A. Laux, and W. Nutt. Terminologische Logiken. *KI*, (3):23–33, 1992.

[2] F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. Research Report RR-91-10, DFKI, Kaiserslautern, Germany, April 1991.

[3] G. E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Proc. of the Second GI Conference on Automata Theory and Formal Languages*, number 33 in LNCS, pages 512–532. Springer, 1975.

[4] M.R. Cutkosky, R.S. Engelmore, R.E. Fikes, M.R. Genesereth, T.R. Gruber, W.S. Mark, J.M. Tenenbaum, and J.C. Weber. PACT: An Experiment in Integrating Concurrent Engineering Systems. *IEEE Computer*, (1):28–37, 1993.

[5] Andreas Dolzmann and Thomas Sturm. REDLOG – Computer Algebra Meets Computer Logic. Technical Report MIP-9603, Universität Passau, Passau, Germany, February 1996.

[6] P. Hanschke. *A Declarative Integration of Terminological, Constraint-Based, Data-driven, and Goal-directed Reasoning*. Dissertation, Universität Kaiserslautern, 1993.

[7] H. Hong. RISC-CLP(Real): Constraint Logic Programming over the real numbers. In *Constraint Logic Programming: Selected Research*. MIT Press, Cambridge, MA, 1993.

[8] ISO. *ISO 10303 Part 105: Integrated Application Ressource: Kinematics*. ISO, 1994.

[9] Gerd Kamp and Holger Wache. CTL – a description logic with expressive concrete domains. Technical report, LKI, 1996.

[10] Daniel Kuokka and Brian Livezey. A Collaborative Parametric Design Agent. In B. Hayes-Roth and R. Korf, editors, *Proc. of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 1, pages 387–393, Seattle, WA, 1994. AAAI Press.

[11] Jean-Louis Lassez. Parametric queries, linear constraints and variable elimination. In A. Miola, editor, *Design and Implementation of Symbolic Computation Systems*, volume 429 of *LNCS*, pages 164–173, Capri, Italy, April 1990. Springer Verlag.

[12] J.G. McGuire, Daniel R. Kuokka, J.C. Weber, J.M. Tenenbaum, T. Gruber, and G. Olsen. SHADE: Technology for Knowledge-Based Collaborative Engineering. *Concurrent Engineering: Research & Applications*, 1(3), 1993.

[13] J.G. McGuire, R.N. Pelavin, J.C. Weber, J.M. Tenenbaum, T. Gruber, and G. Olsen. SHADE: A Medium for Sharing Design Knowledge among Engineering Tools. Annual report, 1992.

[14] J.G. McGuire, R.N. Pelavin, J.C. Weber, J.M. Tenenbaum, T. Gruber, and G. Olsen. SHADE: Knowledge-Based Technology for the Re-Engineering Problem. Annual report, 1993.

[15] P. F. Patel-Schneider and B. Swartout. Description Logic Specification from the KRSS Effort. November 1993.

[16] F. Reuleaux. *The Kinematics of machinery - outlines of a theory of machines*. Macmillan & Co, New York, NY, 1876.

[17] A. Tarski. A Decision Method for Elementary Algebra and Geometry. In *Collected Works of A. Tarski*, volume 3, pages 300–364.

[18] Volker Weispfenning. Applying Quantifier Elimination to Problems in Simulation and Optimization. Technical Report MIP-9607, Universität Passau, Passau, Germany, April 1996.