Appeared in: Description Logics' 96, Boston,MA

Using Description Logics for Consistency-based Diagnosis

Gerd Kamp
Universität Hamburg
Vogt-Kölln-Str.30
22527 Hamburg
kamp@informatik.uni-hamburg.de

Holger Wache
Universität Bremen
P.O. Box 330 440
28334 Bremen
wache@informatik.uni-bremen.de

Abstract

Using quantitative models of simple mechanisms as an example domain, we show how the basic principles of consistency-based diagnosis can be implemented using description logics with expressive concrete domains. In addition, description logics provide the inference services necessary to organize and validate model libraries, an aspect that is neglected in consistency-based diagnosis until now.

1 Description Logics for Diagnosis

Consistency-based diagnosis[1] [7; 10] is a method for the diagnosis of technical devices based on descriptions of the correct and faulty behavior of components. Therefore, for representing, simulating and diagnosing the simple bike drive train in Fig.1 at least the following knowledge must be represented 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.[2]
2. The *structure* of assemblies, e.g the kinematic structure of the mechanisms using kinematic pairs.
3. *Physical laws* like $M = r \times F$ and *constraints* imposed on the attributes like $F \geq 0$ 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 ($M_1 = M_2$) in a rotational pair.

Based on this representation *diagnosis* is mainly the task of identifying the models of behavior that are consistent with a set of observed parameter values. This is most often done by *simulating* hypothetized behaviors and comparing the results with the observed values.

[1] Also called model-based diagnosis

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

In this paper we argue that description logics (DL) are well suited for consistency-based diagnosis. They provide a rich and expressive representation language, that is able to fulfill the representation needs. The TBox contains the descriptions of the different component types and the models of correct and faulty behavior. Concrete devices are modeled as a set of assertions within the ABox. Furthermore the powerful T- and ABox inference services such as *consistency*, *object classification*[3] and *classification* can be used for simulation and diagnosis. Moreover, for certain languages (e.g. $\mathcal{ALCF}(\mathcal{D})$) sound and complete algorithms for these inferences could be given (see e.g. [3]).

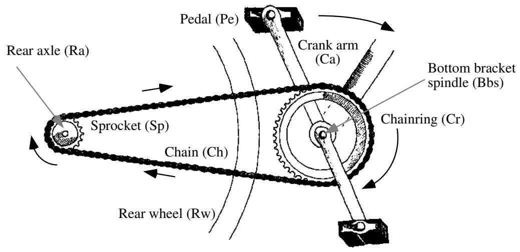

Figure 1: A simple bike drive train

Despite these facts there are no consistency-based diagnostic systems that use description logics, at least to our knowledge. The main reason for this is the lack of expressive concrete domains in current DL systems. Since we have to deal with numeric parameter values, we need a description logic that is able to handle concrete domains. In order to describe physical laws and behavioral models we need at least a concrete domain that is able to reason with linear systems of inequalities between polynomials[4], in the TBox as well as in the ABox.

2 Concrete Domains – State of the Art

Baader and Hanschke [1; 3] have shown that it is theoretically possible to integrate systems of nonlinear polynomi-

[3] Also known as realization.

[4] $M = r \times F$ is a quadratic polynomial but becomes linear when one of the variables is known.

als into the T- and ABox reasoning services. They developed a scheme for integrating concrete domains into description languages and showed that the decidability of the resulting description language $\mathcal{ALCF}(\mathcal{D})$ is preserved for concrete domains that fulfill certain restrictions (so called *admissible* concrete domains).

By using the decidability of the theory of the elementary algebra over the reals [11], it can easily be shown that systems of inequations between arbitrary polynomials and systems of inequalities between linear polynomials are admissible concrete domains [6]. But what is the status of concrete domains in implemented systems?

KRSS provides a concrete part of the domain, specified as the rationals and strings over some alphabet of size at least 2. We restrict our analysis to the numeric domain.

	CORE	LOOM	CLASSIC	KRIS	TAXON
Basetype	$\mathbb{Q},\mathbb{Z}$	$\mathbb{R},\mathbb{Z}$	$\mathbb{R}$	$\mathbb{N}$	$\mathbb{Q}$
Terms					
a	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
x	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
$(x+a)$					
$(x+y)$					
$(a \cdot x)$					
$(x \cdot y)$					
Formulae					
$(\alpha < \beta)$		$\oplus$		$\oplus$	$\oplus$
$(\alpha > \beta)$		$\oplus$		$\oplus$	$\oplus$
$(\alpha = \beta)$	$\odot$	$\oplus$	$\odot$	$\oplus$	$\oplus$
$(\alpha \neq \beta)$		$\oplus$		$\oplus$	$\oplus$
$(\alpha \leq \beta)$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
$(\alpha \geq \beta)$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
Operator					
$\neg\varphi$		$\otimes$		$\otimes$	$\otimes$
$(\varphi \vee \psi)$		$\otimes$		$\otimes$	$\otimes$
$(\varphi \wedge \psi)$	$\otimes$	$\otimes$	$\otimes$	$\otimes$	$\otimes$
Quantifier					
$(\forall x \varphi)$	$\otimes$	$\otimes$	$\otimes$	$\otimes$	$\otimes$
$(\exists x \varphi)$		$\otimes$			$\otimes$

$\oplus$ explicitly contained $\odot$ implicitly expressible
$\otimes$ via the abstract domain

Table 1: Expressiveness: Linear Sentences

Table 1 summarizes the systems' capabilities to handle systems of (in)equations between polynomials.

The first and foremost observation is the *complete lack of any function symbols*. Therefore, current terminological systems can at best handle arbitrary comparisons between attributes or attributes and numbers. Only LOOM defines a relation $+$, but it cannot be used within concept definitions. Therefore it is impossible to define even the simplest equations between univariate linear polynomials like $x = y + 1$ or $d = 2 \cdot r$. Not to mention linear multivariate polynomials like $u = 2 \cdot (l + w)$ or nonlinear polynomials like $M = F \cdot r$.

KRSS and CLASSIC show another notable limitation. Unlike the other systems they provide only $\geq$ and $\leq$ as comparison operators, and one term of the equation is restricted to be a constant. This makes it even impossible to define a square as a special rectangle with $l = w$. One has to use the abstract equality predicate equal in order to describe this. Furthermore, no system implements logical operators or quantifiers within the concrete domain. Instead, they all rely on the respective operators of the abstract domain, resulting in obvious performance hits. Since KRSS and CLASSIC only provide and in the abstract domain, it is impossible to define the missing comparison operator $<$ in these systems.

The only way to circumvent the above limitations in some way is via constructs (satisfies, test-h/c) that allow to include arbitrary Lisp functions into concept descriptions[5]. Obviously this constructs cannot be used at all within the TBox reasoning services. Thus, the handling of concrete domain expressions as a black box may result in incomplete or wrong classifications. In Section 7 we will see that classification based on concrete domain expressions is indeed an interesting TBox inference. W.r.t. the ABox reasoning they have the major disadvantage of being lisp functions and not constraints.

All this shows that current implementations of description logics do not provide the mechanisms we need in order to fulfill the representation needs described in Section 1.

3 Expressive concrete domains

There are two possibilities for implementing the concrete domain extension of a terminological system: providing a generic interface to algorithms realizing a concrete domain[6] or hardwiring the integration of specific algorithms[7]. A generic interface makes it easy to provide multiple concrete domains, resulting in a modular system architecture. Thus one can select the proper algorithms needed to fulfill the requirements of the specific application, and couple them with the description logic through the generic concrete domain interface.

For example, a large number of algorithms for solving systems of algebraic (in)equations are and have been developed. Furthermore, a couple of this algorithms have been incorporated into CLP($\mathcal{R}$)–systems (e.g. [5]). Using a CLP($\mathcal{R}$)–system as the concrete domain solver, we participate from the progress made in this area through simply exchanging the CLP($\mathcal{R}$)–system at the interface.

3.1 Concrete predicates

Thus, we implemented CTL [6], a system based on TAXON and the description logic $\mathcal{ALCF}(\mathcal{D})$ which realizes a generic concrete domain interface. CTL implements the KRSS syntax, which we had to be extended in the following way (for details see [6]): *Predicate terms* P describe concrete predicates. They are either a predicate name PN or a list ($\langle$name$_\mathcal{D}\rangle$ $(x_1 \ldots x_n)$ $\langle$expr$_\mathcal{D}\rangle$) consisting of a domain identifier $\langle$name$_\mathcal{D}\rangle$ and a list of variables $x_1 \ldots x_n$. The expression $\langle$expr$_\mathcal{D}\rangle$ actually defines the concrete predicate. (define-

[5] KRSS , LOOM , and CLASSIC provide such constructs.

[6] This is mainly the task of deciding if a finite conjunction of concrete domain expressions is satisfiable.

[7] The built-in concrete domains in CLASSIC , LOOM and KRIS are more or less examples for this hard wired integration.

constraint $PN\ P$) assigns a name PN to a predicate term P. Further (constrain $R_1\ \ldots\ R_nP$) is an additional concept term operator with the following semantics: (constrain $R_1\ \ldots\ R_nP)^{\mathcal{I}} = \{a \mid \exists b_1, \ldots, b_n((a, b_1) \in R_1^{\mathcal{I}} \wedge \cdots \wedge (a, b_n) \in R_n^{\mathcal{I}} \wedge (b_1, \ldots, b_n) \in P^{\mathcal{I}})\}$.

Since CTL is based on the plain tableaux calculus using the rules described in [3] only the respective concrete domain rules had be implemented appropriately in order to realize the generic concrete domain interface.

3.2 Generating models and computing parameter restrictions

Description logic systems based on a tableaux method explicitly generate a model in the course of the consistency test. Normally only the result of the consistency test is presented to the user and the constructed model is more or less thrown away. In technical domains and especially in our application (but also e.g. in configuration) the calculated model is the main object of interest. Therefore we realized a function (show-model) within CTL that returns the model M for a consistent ABox.

This schema can be extended to concrete domains by using quantifier elimination techniques from computer algebra[8]. A subtask of the consistency test for an ABox, is testing whether a conjunction of concrete predicates is satisfiable, i.e. checking whether the sentence $S = \exists x_1 \ldots \exists x_k(p_1 \wedge \cdots \wedge p_n)$ is true. Tarskis basic idea for deciding the theory of the elementary algebra was to transform an arbitrary sentence into an equivalent quantifier free one $\hat{S}$ [11]. The validity of this sentence is then easy to check. Over the years quantifier elimination techniques have been vastly improved, more efficient general algorithms have been devised (e.g. Cylindrical Algebraic Decomposition (CAD) [2]) and further improved (e.g. PCAD [4]) and specialized algorithms for subsets of the theory have been found and realized. Most CLP($\mathcal{R}$)-systems make use of the Fourier-Motzkin algorithm for quantifier elimination over linear systems of inequalities (see e.g. [8]).

Besides checking the validity of a sentence, quantifier elimination as a generic method can be used to compute the restrictions imposed on the parameters of the model M:

1. Let $X = x_1, \ldots, x_n$ a list of parameters in M.
2. For $x_i \in X$
   (a) Eliminate all quantifiers but x_i from S. I.e. transform S into a sentence $S_i = \exists x_i R_i, R_i$ quantifier free.
   (b) Transform S_i in disjunctive normal form, i.e. transform S_i into $\hat{S}_i = \mathrm{dnf}(S_i)$.
   (c) Collect the matrix R_i of $\hat{s}_i$.
3. Return the list $R = R_1, \ldots R_n$ of collected matrices R_i.

[8]this is called variable elimination or projection in the field of CLP($\mathcal{R}$)

Since R is a list of the admissible values for each parameter, the computed restrictions can then be delivered together with the generated model.

4 Representation

As we have seen in Section 1, we must be able to describe the different component types of a mechanism as well the kinematic structure. In kinematics this is normally done with links and kinematic pairs [9]. In order to describe 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.2a defines a link 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 further specifying them.

In order to describe the 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 degrees of freedom of the relative motion of the links and the type of connection different types of pairs can be identified. In the terminology shown in Fig.2b we restrict us to the description of pairs between two rotational-links (rot-pair) and pairs between one rotational-link and one tension-link (rot-tension-pair). Note that we are able to describe rot-tension-pair via an or construct. In addition to the component types and the structure of the device, descriptions of the correct as well as the different faulty behavior models are needed for the consistency-based diagnosis as well as the simulation of the device. The terminology in Fig.2c describes the correct behavior of rotational-pairs and rot-tension-pairs. Whereas the torque is propagated in ok-rot-pair, ok-rot-tension-pair propagates the force.

The terminology in Fig.2d 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. 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.2e. 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 limit between strong and weak slipping.

5 Simulation

Simulation is based on the following procedure:

a) Representation: Links

```
(define-primitive-attribute link.force Top)
(define-primitive-concept link
  (and Top (some link.force (minimum 0))))
(define-primitive-concept tension-link link)
(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)))
```

b) Representation: Kinematic Pairs

```
(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)))))
```

c) Representation: Normal behaviour

```
(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)))
```

d) Representation: Faulty behaviour (I)

```
(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) (?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)))))
```

e) Representation: Faulty behavior (II)

```
(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)))
```

f) Simulation: Setting up a drive train

```
(define-distinct-individual crankarm-1)
…
(define-distinct-individual rear-wheel-1)
(define-distinct-individual pair1)
…
(define-distinct-individual pair6)
(state (and
  (related pair1 crankarm-1 pair.link1)
  (related pair1 bottom-bracket-spindle-1
                 pair.link2)
  …
  (related pair6 rear-axle-1 pair.link1)
  (related pair6 rear-wheel-1 pair.link2)))
(state (and
  (instance crankarm-1 rotational-link)
  …
  (instance rear-wheel-1 rotational-link)
  (instance pair1 kinematic-pair)
  …
  (instance pair6 kinematic-pair)))
```

g) Simulation: Asserting parameters and behaviours

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

h) Diagnosis: Setting up a rotational pair:

```
(define-distinct-individual rot-pair1)
(define-distinct-individual link1.1)
(define-distinct-individual link1.2)
(state (and
  (instance rot-pair1 rot-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)))
```

i) Diagnosis: Different parameter sets

1) ok-rot-pair:

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

2) broken-rot-pair

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

3) weak-slipping-rot-pair

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

Figure 2: The Krss code

1. Description of the device: Instantiating, relating the instances to primitive concepts and building up the device structure by relating the instances.
2. Determination of the models of behavior for the different device parts.
3. Assertion of some parameter restrictions.
4. Checking the consistency of the ABox. The values and restrictions of additional parameters are calculated during the model-generating process. Furthermore, additional instances may be generated.
5. Presentation of the generated model and parameter restrictions. Eventually backtracking in order to calculate another model.

It is clear that the assertion of behavior models and parameter restrictions can be freely intermixed. In addition, the consistency test and therefore the model generation can be triggered after every step. In the following we will give a brief overview how simulation works, using the drive train depicted in Fig.1. First the structure of the drive train has to be described: Instances have to be instantiated, primitive concepts for this instances must be given and the links and the kinematic pairs must be related (Fig.2f).

In a second step (Fig.2g) we give the values of the radii of the crankarm-1, chainring-1, sprocket-1 and rear-wheel-1 and the force that is applied to the crankarm as starting values[9]. We then state that all components expose their normal behavior. Finally, we give a value for the force applied to crankarm-1. Now the values for nearly all missing parameter values can be calculated (see Figure 3). W.r.t. the bottom-bracket-spindle-1 and the rear-axle-1 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 due to the law of torque that holds for intact rotation-pairs.

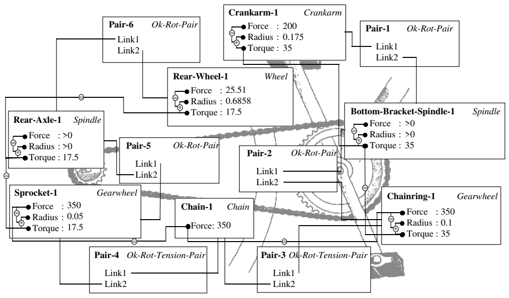

Figure 3: **The generated model and parameter restrictions**

Arbitrary other parameter restrictions as well as an arbitrary selection of behaviors are possible, but due to the lack of space we can give no further examples.

6 Diagnosis

Diagnosis is supported in the following way:

1. Description of the structure of the device.
2. Assertion of the observed parameter values and (re)realization of the affected components. This gives an upper bound for the behaviors that are consistent with the observed values.
3. Weak realization in addition gives a lower bound for the possible models of behavior and therefore hints which parameter values should be determined next. This values are than asserted, the components (re)realized etc.

[9]The values are given in Meter and Newton

We illustrate this method, using a single rotational-pair as an example. First Fig.2h shows how to describe a rotational pair within the ABox. Fig.2i then shows how different sets of parameter values lead to different diagnoses. While in the first variant identic torque values lead to ok-rot-pair, giving a zero value for the torque of one link and a non-zero value for the for force of the other link suffice to recognize it as a broken-rot-pair. This is possible because the radius is restricted to be strictly positive. Using the physical law it can be concluded that the torque of the second link is strictly positive which leads to the recognition as a broken-rot-pair. In the third variant the pair is realized as a weak-slipping-rot-pair, due to the calculated ratio of the torques.

7 Model Libraries

Fig.4 shows the concept hierarchy after classification of the above terminologies. In this section we will show how classification can be used for the organization and maintenance of model libraries. The following observations are important w.r.t this aspect:

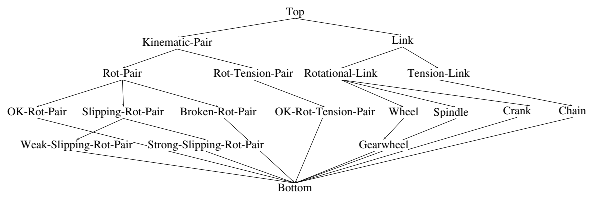

Figure 4: The classified TBox

All concept definitions are different from bottom. Therefore all definitions are satisfiable. This guarantees that no model of behavior is mistakenly defined in a way that there exists no parameter combination that leads to this behavior.

All concept definitions are distinct from each other. This means that there are no two models of behavior that are equivalent, something that could easily happen when two model libraries are merged.

The strong and weak slipping pairs in Fig.2e are modeled as specializations of rot-pair and not of slipping-rot-pair. Situations like this are very likely if different people are simultaneously developing models of behavior, or when the model libraries are complex. The classification service detected 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 accidental in a sense that it is caused by some error or laxness in the description of the models of behavior. Such Errors are very likely 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[10] in CTL, we can guarantee that all missing and accidental subsumption relations in the model library are detected.

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

8 Summary and outlook

Description logics with expressive concrete domains are well suited for consistency-based diagnosis and simulation of technical devices. Additionally they provide the inferences that are needed in order to organize and maintain large model libraries. Actual work focuses on interfacing to a computer algebra system for quantifier elimination over quadratic sentences [12]. Further work concentrates on concrete domains over other base types, e.g. using CLP(FD) systems for describing qualitative models. Finally, we explore other application areas such as configuration and intelligent retrieval from parts catalogs etc.

References

[1] F. Baader and P. Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. Research Report, DFKI, Kaiserslautern, Germany, 1991.

[2] 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*, Springer, 1975.

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

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

[5] Joxan Jaffar, Spiro Michaylov, Peter J. Stuckey, and Roland H. C. Yap. The CLP(R) Language and System. Technical report, 1987.

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

[7] J. de Kleer and B. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32, 1987.

[8] Jean-Louis Lassez. Parametric queries, linear constraints and variable elimination. In *Proc. DISCO-90* 1990.

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

[10] P. Struss and O. Dressler. Physical Negation - Integrating Fault Models into the General Diagnostic Engine. In *Proc. IJCAI-89*, 1989.

[11] A. Tarski. A Decision Method for Elementary Algebra and Geometry. In *Collected Works of A. Tarski*.

[12] Volker Weispfenning. Applying Quantifier Elimination to Problems in Simulation and optimization. Technical Report, Universität Passau, 1996.