KINEMATIC CHAIN SUBSTITUTION FOR GEOMETRIC CONSTRAINT SOLVING*

Gül Ogan
IWF
Swiss Federal Institute of Technology
CH-8092 Zürich, Switzerland
email: ogan@iwf.bepr.ethz.ch

Felix J. Metzger
IWF
Swiss Federal Institute of Technology
CH-8092 Zürich, Switzerland
email: metzger@iwf.bepr.ethz.ch

Bernhard J. Seybold
IfTI
Swiss Federal Institute of Technology
CH-8092 Zürich, Switzerland
email: seybold@inf.ethz.ch

Max Engeli
IWF
Swiss Federal Institute of Technology
CH-8092 Zürich, Switzerland
email: engeli@iwf.bepr.ethz.ch

ABSTRACT

Assembly modelling is an important task within the design process for mechanical engineering. Constraint-based modelling is an adequate approach to this task. Kinematic mechanisms often have multiple interacting loops which are difficult to solve by a geometric constraint solver. In this paper, we introduce *kinematic chain substitution*, which is an elegant method to solve loops. Kinematic chains are open paths of simply connected components in which — unlike rigid chains — relative mobility between their components is possible. For *kinematic chain substitution*, the chain between the first and last component is substituted temporarily with one arc, by eliminating the other components in between. Our algorithm, called *Concatenate*, calculates the relative mobility restriction between the first and last component of a kinematic chain with 3 components. By repeatedly applying *Concatenate*, a *kinematic chain substitution* for chains with any number of components is possible. The main application of this approach is to solve loops. In addition, *Concatenate* can be used to visualise the relative mobility restriction between any two components, which is induced by a particular chain. Our approach is an advantageous alternative with respect to *locus analysis* presented by KRAMER, because it is more flexible and has a more general application scope.

Keywords geometric constraint solving, loops, kinematic chains

*Research supported in part by Swiss Government, KTI 2726.1.

1 INTRODUCTION

Assembly modelling, which deals with the joining of separate rigid bodies in 3D-space, is an important task within the design process for mechanical engineering. Constraint based modelling is a modern approach (for overview cf. [AM96]), which is adequate for this application as shown by other work [Kra92, Bru94, AKC96]. By using geometric constraints, the mobility restrictions can be described declaratively, which allows to separate the modelling part and the solving part of the problem. Declarative description is a more natural and easier way to capture and store the engineer's intent. The design can easily be changed by changing the constraints [ZGV92, Bru94]. The engineer can analyse the kinematic structure and can detect design or mounting problems early by performing simulations and by getting feedback about redundancy or bad numerical conditions of the design. Mechanisms, in particular those in the kinematic domain, can have ***multiple interacting loops*** corresponding to coupled equations, which must be solved simultaneously. Often this equations are non-linear and highly coupled, therefore solving such loops is a challenging task for a geometric constraint solver.

In literature, many publications in the area of geometric constraint solving can be found. Some of them are the following: POPPLESTONE at al. [PAB80] use symbolic expressions to describe mechanical assemblies. Their solver is able to handle a number of interesting mechanical assembly problems, but some other mechanisms with multiple interacting loops remain too complex to solve. ROMDHANE [Rom92] can handle loops with three components called dyads. His method depends on the ability to decompose the mechanism into dyads and to apply a known analytical solution for each of those dyads. However, he considers only planar mechanisms with revolute and prismatic joints. OWEN [Owe91, Owe96] describes an algorithm which computes the solution formula for ruler-compass constructible[1] configurations only, using purely algebraic methods. FUDOS and HOFFMAN [FH97] describe a graph-constructive approach to solve systems of constraints. This approach handles well-constrained, over-constrained and under-constrained configurations efficiently. However, they restrict themselves to points and lines with pairwise distance and angle constraints in 2D-space. BRÜDERLIN [Brü93] describes a theoretical framework of a geometric constraint solver based on rewrite rules which are directly derived from the axioms of Euclidian geometry. He also considers only ruler-compass constructible configurations. KRAMER [Kra92] describes a degree-of-freedom-analysis method to solve kinematic linkages in 3D-space, in which he uses *locus analysis* to solve loops. However, without the extended loci, his method can only handle loops with three components. A short overview of the first version of our constraint solver, which is partly based on KRAMER's approach, is given in [SMO+97].

[1] Configurations which could in principle be constructed on a drawing board using standard drafting construction techniques. They can be solved using only algebraic operations and square root.

In this paper, we explain a ***kinematic chain substitution algorithm*** called ***Concatenate***, which can be used to solve loops. It can also be used to analyse the relative mobility between two components induced by a particular kinematic chain. It is an alternative approach to ***locus analysis*** of KRAMER and is more flexible as well as more general. Our approach is not restricted to loops with three components. For our application the components are rigid bodies in 3D-space which are connected with binary geometric constraints. They are building a non-linear constraint solving problem. The variables describe the position and orientation of the components. The idea behind Concatenate however is more general, it can also be used for non-geometric problems which have a similar structure.

In Section 2, we give a short summary of our solver. In Section 3 we explain the idea of the kinematic chain substitution. In Section 4, we introduce the *Concatenate*, which is compared with the locus analysis in Section 5. Conclusions are given in Section 6.

2 SOLVER OVERVIEW

The ***assembly*** of rigid bodies in 3D-space can be modelled by a ***geometric constraint graph***, in which the nodes are ***components*** and the edges are ***binary geometric constraints***, which restrict the relative mobility between its two adjacent components. This graph can be converted into a ***state graph***, where edges are ***states***, describing the relative mobility between the nodes. These nodes are also called ***components*** for simplicity. These components do not correspond one-to-one to those in the constraint graph, but they can be composed of more than one such component. This happens whenever there is no relative mobility left between components. The conversion from the constraint graph to the state graph is done by the ***geometric constraint solver***.

Syntax	Semantic
$Coincident(P_1, P_2)$	point P_1 coincides with point P_2
$In\text{-}line(P_1, L_2)$	point P_1 lies on line L_2
$In\text{-}plane(P_1, Pl_2)$	point P_1 lies on plane Pl_2
$Identical(M_1, M_2)$	marker M_1 coincides with marker M_2
$Distance(P_1, P_2, d)$	distance between point P_1 and point P_2 is d
$Offset\text{-}x(T_1, T_2)$	thrihedral T_1 coincides thrihedral T_2
$Parallel\text{-}z(V_1, V_2)$	vector V_1 and vector V_2 are parallel
$Offset\text{-}z(V_1, V_2, \alpha)$	angle between vector V_1 and vector V_2 is α

Figure 1: Basic constraint types, except the semi-algebraic ones

Our constraint solver is implemented as a global exploration algorithm based on local solving algorithms. There are two kinds of local solving algorithms:

- those which solve 2-component problems, i.e. two components with constraints between them: ***constraint addition*** (adding a new constraint to a given state) and ***state combination*** (combining two states between the same components into one state).
- one algorithm which handles 3-component problems, i.e. three components which have constraints between them, in order to simplify n-component problems: ***Concatenate*** (***cat***).

The basic constraint types of our solver are shown in Figure 1. More details of our solver and the algorithms can be found in [SMO+97] and [SOMS97].

3 KINEMATIC CHAIN SUBSTITUTION: CONCATENATE

A ***kinematic chain*** is an open path of two or more simply connected components where — unlike rigid chains — it is possible to have relative mobility between all its components. In a ***kinematic chain substitution***, a chain is substituted by another one with two components, by eliminating the components between the first and last

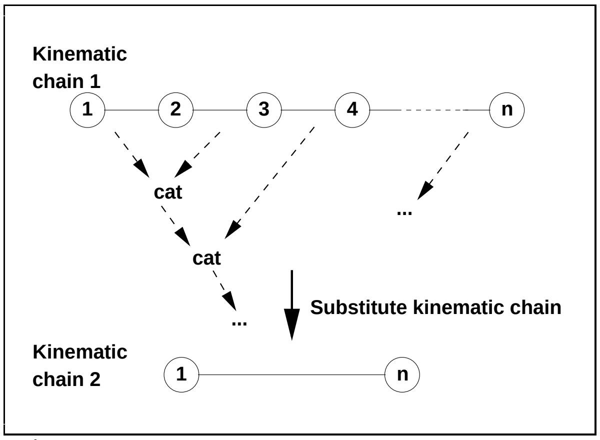

Figure 2: Kinematic chain substitution by repeated application of the concatenate routine.

one. The algorithm which calculates the relative mobility restriction between the first and last component of a kinematic chain with 3 components is called ***Concatenate***. With *Concatenate*, a kinematic chain of three components can be replaced by a chain of two components. By repeatedly applying this algorithm, a kinematic chain substitution for kinematic chains with more than three components is possible (cf. Figure 2).

There are two main applications of *Concatenate*:

- to solve a loop problem by converting it into a 2-component problem. For this application, the result of *Concatenate* is added to the graph. In a loop, one can choose two components and substitute each kinematic chain between these two components, one after the other, by repeatedly applying *Concatenate*. After each step the result is combined with all preceding ones. This process is stopped, if the appearing 2-component problem has no relative mobility left after solving, such that it can be composed into one component. In this context, substitution is temporary. The original kinematic chain remains in the graph. If two components can be composed into one component, the remaining problem is simpler. It is of the same problem class as the original one and can be solved in the same way. This strategy can be repeated, and possibly the whole problem can be solved this way.
- to visualise and to analyse the relative mobility restriction between any two components, which is induced by a particular kinematic chain between them. For this application, the result of *Concatenate* substitutes the original kinematic chain. There are real life mechanisms that form an open kinematic chain. (cf. Figure 4).

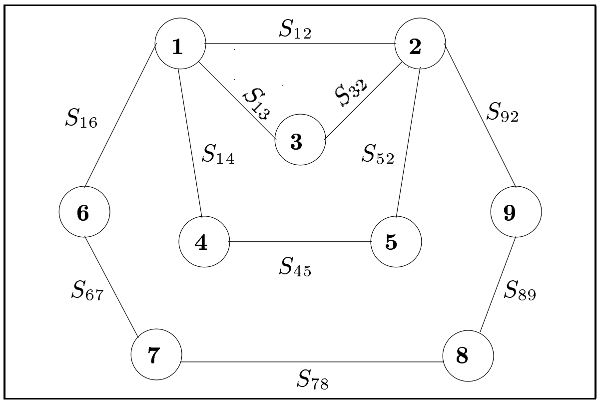

Figure 3: There are four different kinematic chains between components 1 and 2: **(a)** S_{12}; **(b)** S_{13}, S_{32}; **(c)** S_{14}, S_{45}, S_{52}; **(d)** S_{16}, S_{67}, S_{78}, S_{89}, S_{92}. The graph represents multiple interacting loops. (S denotes for State)

An example for the first application is given in Figure 3. One step of the solving process for this example can be as follows: *Concatenate* is applied for chain (b). The combination of the result and the chain (a) together form a 2-component problem with the components 1 and 2. If after solving this problem, with 2-component solver routines, there is no relative mobility left between 1 and 2, then they can be composed into one component called 1-2. Then the two components 3 and 1-2 together form again a 2-component problem, which can be handled with 2-component solver routines. If after solving this problem, there is no relative mobility left between the component 3 and 1-2, they can be composed into one component 1-2-3. Now the loop consisting of the components 1, 2 and 3 is solved. The result is the component 1-2-3.

4 DEFINITION OF CONCATENATE

Given the components i, j and k, the state S_{ij} between i and j, and S_{jk} between j and k, *Concatenate* calculates the state S_{ik}, which describes that restriction of the relative mobility between the component i and k, which is induced by the kinematic chain.

4.1 A Simple Mechanical Example

One simple example is shown in Figure 5(A). The relative mobility between the components i and j is a translation along a line. The relative mobility between j and k is a translation along another line,

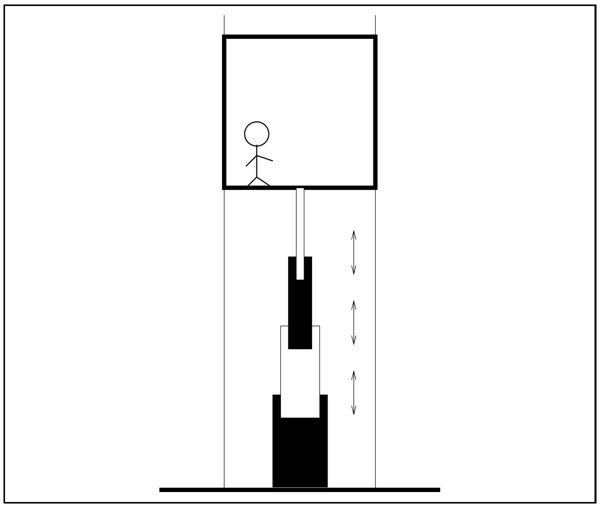

Figure 4: A hydraulic elevator, which uses the telescope mechanism, as an example of an open kinematic chain. *Concatenate* can calculate the relative mobility between the ground and the elevator. If we ignore the rotational mobility, it is a chain composed of a repetition of the mechanism in Figure 5(B).

which is not collinear to the first line. This results in a relative mobility between i and k, which is a translation on a plane. This relative mobility information is induced by the chain (i, j, k).

A special case occurs, if the two lines are collinear. Then the result is a translation along a line, instead of a translation on a plane. (cf. Figure 5(B)).

4.2 Formalism

A state between the components i and j is represented by a conjunction of constraints between them. For a given state, a list of constraints $List_{ij}$ can always be found such that the conjunction of its constraints defines the state. This list can simply be the list of all constraints which have been given as input between i and j, or it can be a simplified, transformed version (a normal form) of this list which is equivalent to the original. Normalisation is performed by local solving algorithms.

As a simple example, consider a state which only allows a translation along a line (similar to state 1 TDOF, 0 RDOF according to KRAMER, [Kra92]). This situation can be represented by the following list of two constraints: $\{$*offset-x*(T_1, T_2), *in-line*$(P_1, L_2)\}$, assuming that the geometric data of the constraints (values of T_1, T_2, P_1, L_2) are chosen appropriately. The syntax of these con-

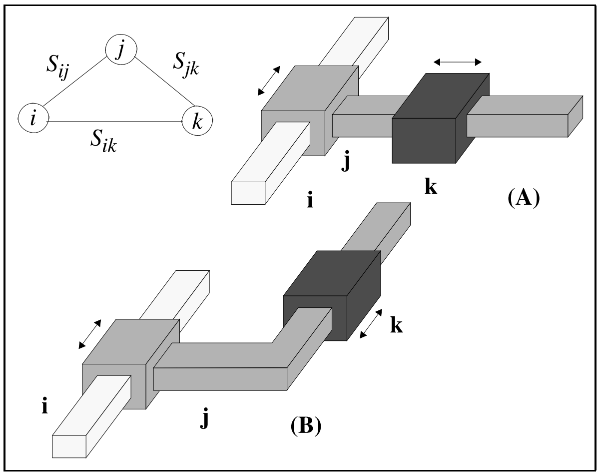

Figure 5: Example (A): The relative mobility between i and k is a translation on a plane. (B): The relative mobility between i and k is a translation parallel to a line.

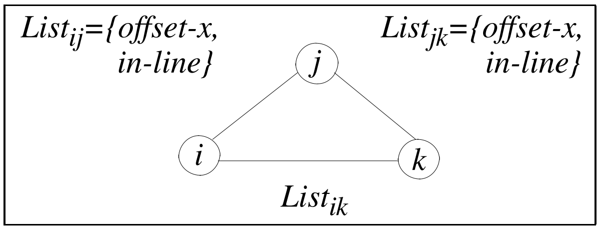

Figure 6: Relative mobility of components represented by normalised constraint lists.

straints is defined in Figure 1, together with the other constraints. *Offset-x* means that a trihedral[2] T_1 on the one component must coincide with a trihedral T_2 on the other component. It reduces the rotational mobility between the two components completely. *In-line*, which indicates that a point P_1 of the one component must lie on a line L_2 of the other component, reduces the translational degrees of freedom to one. This is the same situation as in Figure 5(A) between the components i and j. To describe the relative mobility as separate translational and rotational degrees of freedom is only possible for special cases, which, however, occur often in mechanical engineering.

For the concatenation of state S_{ij} and state S_{jk}, consider constraint list $List_{ij}$, and $List_{jk}$ which are equivalent to the states. *Concatenate* finds a new constraint list $List_{ik}$, equivalent to the state between i and k, which is the result of the concatenation (cf. Figure 6). It is important that $List_{ij}$ and $List_{jk}$ are normalised in advance, firstly to avoid combinatoric explosion and secondly because otherwise Concatenate would detect less mobility restrictions.

[2] three right-handed, normalised vectors, mutually orthogonal

Concatenate utilises all applicable rewriting rules to $List_{ij}$ and $List_{jk}$, explained in Section 4.3. l_{ij} denotes a sublist of $List_{ij}$, and l_{jk} a sublist of $List_{jk}$. For a pair of sublists (l_{ij}, l_{jk}) satisfying the preconditions for a specific rewriting rule, this rule is applied and results in a sublist l_{ik}, which is added to $List_{ik}$. $List_{ik}$ is empty at the beginning.

The result of *Concatenate*, $List_{ik}$, can still be an empty list. This indicates that the algorithm has not found a restriction of the mobility between the components i and k, either due to the nonexistence of such a restriction, (e.g. if one of the input lists is empty), or due to the inability of *Concatenate* to detect it.

4.3 Rewriting Rules

In the local solver, there are generally two kinds of rewriting rules: 2-component-rewriting-rules, and 3-component-rewriting-rules, which both are applied on the basis of constraint lists.

The 2-component-rewriting rules are rules for constraints between two components. Consider a conjunction of several constraints between two components. In a conjunction of constraints, the case in which there is only one constraint shall be included. After having applied a rule, we get a conjunction of constraints between these two components. The applicability of these rules depends on the geometric data of the constraints. These rules serve to normalise the constraint list when applied appropriately by a normalising strategy.

A simple example: If there is an *offset-x* between two components, both trihedrals of the *offset-x* can be rotated collectively, such that one of the trihedrals coincides with a given third one. The result is another *offset-x* equivalent to the original *offset-x*.

The 3-component-rewriting rules are defined between three components. There is a conjunction of constraints between the components i and j, and a conjunction of constraints between the components j and k. After having applied a rule, we get a conjunction of constraints between components i and k. The applicability of these rules depends on the geometric data of the constraints as well. While 3-component-rewriting rules are only used for the *Concatenate* routine, 2-component-rewriting rules are used also in other local solving algorithms. The 2-component-rewriting rules can be used to transform a conjunction of constraints, in order to satisfy preconditions of the 3-component-rewriting rules.

Semi-algebraic constraints are constraints defining inequalities. They are not considered in our *Concatenate* implementation, because they do not reduce the dimension of the mobility (dimension of the solution set). But they only serve to choose between discrete freedoms and they are too expensive to be implemented within *Concatenate*.

Given the set of basic constraint types of our solver (about 10 constraint types, cf. Figure 1), we have less than 30 3-component-rewriting rules of the following form:

- Rewriting-Rule(l_{ij}, $l_{jk} \rightarrow l_{ik}$)

We believe that our *Concatenate* implementation can detect all results occurring in practical applications — for our constraint types except the semi-algebraic ones — which reduce the mobility between the first and last component, induced by a chain between them.

Concatenate($List_{ij}$,$List_{jk}$) performs up to *O(n*m)* rule set application operations, where n is the length of $List_{ij}$ and m is the length of $List_{jk}$. Every such operation runs in constant time. Because the input lists are normalised before applying Concatenate, the typical length of the lists does not exceed five.

4.4 Example of a 3-component-rewriting Rule

We give a simplified example for a 3-component-rewriting rule, which corresponds to the case in Figure 5(B). The components i, j and k with the states S_{ij} and S_{jk} are given. The normalised constraint lists are:

- $List_{ij}$ = {Offset-x(T_i,T_j^1), In-line(P_j,L_i)}
- $List_{jk}$ = {Offset-x(T_j^2,T_k), In-line(P_k,L_j)}

The rule, we want to apply, is:

- {Offset-x, In-line}, {Offset-x, In-line} $\rightarrow$ {Offset-x, In-line}

This rule is given in Algorithm 1. After having applied the rule to our lists, assuming the geometric data are appropriate, we get the following list:

- $List_{ik}$ = {Offset-x(T_i, T_k), In-line(P_k, L_i)}

Algorithm 1
{Offset-x, In-line},{Offset-x, In-line} $\rightarrow$ {Offset-x, In-line}

Require: existence of:
Offset-x(T_i, T_j^1) and In-line(P_j, L_i),
Offset-x(T_j^2, T_k) and In-line(P_k, L_j)
and satisfaction of:
L_i collinear to L_j
1: use 2-component rewriting rules such that P_j lies on L_j
2: use 2-component rewriting rules such that T_j^1 and T_j^2 are equal
3: $List_{ik} \leftarrow List_{ik}$ + In-line(P_k, L_i)
4: $List_{ik} \leftarrow List_{ik}$ + Offset-x(T_i, T_k)
5: **return**

4.5 Properties of Concatenate

The Concatenate algorithm has the following properties:

- If S_{ij}, and S_{jk} do not correspond to illegal situations (i.e. constraints contradictions, which means there is no solution), the result state S_{ik} will correspond to a legal situation.
- In general, the result of *Concatenate* is not normalised, therefore 2-component solving algorithms must be applied in order to normalise the result.
- To solve the global problem for a chain (i, j, k), the result of *Concatenate*, state (i, k) is added to the graph. As explained before, this state can be described as a conjunction of constraints. These are constraints induced by the constraints in the chain (i, j, k) which are still in the graph. Therefore for the global problem (whole graph) these constraints are redundant. This fact must be taken into account for the calculations of the redundancy information.
- *Concatenate* allows to describe a 3-component kinematic chain as a 2-component kinematic chain, and therefore as a 2-component problem. Assuming that only the relative mobility between the first and last component in the kinematic chain is of interest, a kinematic chain with many components can be described as a 2-component problem by repeatedly applying *Concatenate* to the kinematic chain.
- *Concatenate* is associative. This means that: *(i cat j) cat k* $\equiv$[3] *i cat (j cat k)* holds.

[3]means that the resulting states describe the same solution set.

- Another conceptual viewpoint of *Concatenate* is that position and orientation of components are described by variables and the binary constraints between the components can be written as equations for them. *Concatenate* then eliminates all variables except those associated with the first and last component in the chain.
- *Concatenate* as implemented in our solver, deals with 3D-space geometric problems and is not restricted to 2D-space geometric problems. The concept behind *Concatenate* is more general and can be applied to other non-linear constraint solving problems, which do not have to be geometric but have a similar structure (structure which can be interpreted as described in the last bullet point).

5 COMPARISON WITH LOCUS ANALYSIS

KRAMER'S *locus tables* and *locus intersection tables* are only sufficient to solve loops with three components. For loops with more components, he has to implement the *extended locus tables* which seems to be a lot of work. For the constraints used in kinematic analysis he provides that only 9 extended point loci types of dimensions two and none of dimension one are needed, and that three dimensional loci are not of interest because they only serve to eliminate some discrete solutions (do not reduce the dimension of the solution set). We believe that with this approach for every bigger loop additional special cases must be considered. His idea bases also on the proof that the largest topologically rigid loop has maximum six components. But there are cases which he calls degenerate, where this is not true.

The result of the *Concatenate* can be seen as the locus of one component relative to the other component. One difference to KRAMER is that we talk about the locus of the whole component (i.e. the position and orientation of the component) whereas KRAMER considers only the loci of points, lines and vectors. Consider three components, a state between the first and second component, as well as a constraint between the second and third component. It is assumed that the first component is not moved. Then the state can be interpreted as the whole locus of the second component relative to the first one. The locus of the third component relative to the second component, according to the given constraint can then always be calculated, because we have the whole locus information for the second component. This is not always possible if only the locus of points, lines and vectors are considered.

Another difference is that KRAMER intersects the loci using some auxiliary geometries, converts the result into a constraint, and treats this constraint with *action analysis* (2-component solver). This implies that the loci intersections needed are done outside the 2-component solver. On the contrary, we describe the locus with constraints, which can be treated directly in the 2-component solver. We do not need auxiliary geometries for that purpose. The necessary loci intersections are done inside the 2-component solver. The input to the *Concatenate* can be seen as two 2-component problems, where one component is common. The result of the *Concatenate* is again a 2-component problem. Therefore *Concatenate* is associative and can be applied repeatedly to a kinematic chain.

6 CONCLUSION

We have explained the *Concatenate* algorithm, which calculates the relative mobility restriction between the first and last component of a kinematic chain when applied repeatedly. By choosing any 2 components and substituting all necessary kinematic chains between them temporarily with the help of *Concatenate*, loops for mechanisms in 3D-space can be solved using only 2-component algorithms. Another useful application of *Concatenate* is to analyse the relative mobility restriction between any two components, which is induced by a chain.

The *Concatenate* algorithm is a significant improvement with respect to *locus analysis* of KRAMER. It is more flexible and powerful, because it enables to work always with the same kind of data structure (*state*), instead of applying two different concepts like *state* and *locus*. Our approach is not restricted by the loop size.

We believe that the concept behind *Concatenate* is general and applicable to many kinds of non-linear constraint solving problems of similar structure, which do not have to be geometric problems. According to this concept, if position and orientation of components are described by variables, the binary constraints between the components can be written as equations. *Concatenate* then eliminates all variables, except those associated with the first and last component in the chain. For all constraints solving problems which can be described in this way (for other problems the variables describe other things instead of position and orientation) an algorithm similar to Concatenate can be used.

We believe that our *Concatenate* implementation can detect all results occurring in practical applications — for our constraint types except the semi-algebraic ones — which reduce the mobility between the first and last component, induced by a chain between them.

Further research topics include:

- Describing a theoretical Framework for Concatenate: We believe that, by formalising all rewriting rules in an appropriate way, we can describe it as a rewriting system and prove properties like completeness.
- Implementation of Concatenate for non-geometric applications.

7 ACKNOWLEDGEMENT

We would like to thank Th. Bühlmann, St. Schmälzle and Th. Schnider for their comments on this paper.

References

[AKC96] R. Anantha, G. Kramer, and R. Crawford. Assembly modelling by geometric constraint satisfaction. *CAD, Elsevier Science Ltd*, 28(9):707–722, 1996.

[AM96] R. Anderl and R. Mendgen. Modelling with constraints: theoretical foundations and application. *CAD*, 28:155–168, 1996.

[Brü93] B.D. Brüderlin. Using geometric rewrite rules for solving geometric problems symbolically. *Theoretical computer science*, 116:291–303, 1993.

[Bru94] W. Brunkhart. Interactive Geometric Constraint Systems. Master's thesis, University of California at Berkeley, May 1994.

[FH97] I. Fudos and C. Hoffman. A Graph-Constructive Approach to Solving Systems of Geometric Constraints. *ACM Transactions on Graphics*, 16(2):179–216, 1997.

[Kra92] G. Kramer. *Solving Geometric Constraint Systems.* MIT Press, Cambridge, Massachusetts, London, England, 1992.

[Owe91] J.C. Owen. Algebraic solution for geometry from dimensional constraints. In *Symp. Solid. Modelling Foundations and CAD/CAM Applications ACM SIGGRAPH*, pages 397–407, 1991.

[Owe96] J.C. Owen. Constraints on simple Geometry in two and tree dimensions. *International Journal of Computational Geometry and Applications*, 6(4):421–434, 1996.

[PAB80] R. Popplestone, P. Ambler, and M. Bellos. An Interpreter for a Language for Describing Assemblies. *Artificial Intelligence*, 14:79–107, 1980.

[Rom92] L. Romdhane. A dyad search algorithm for solving planar linkages using the dyad method. *Design Engineering: Mechanism Design and Synthesis*, 46:49–54, 1992.

[SMO+97] B. Seybold, F. Metzger, G. Ogan, J. Bathelt, F. Collenberg, J. Taiber, K. Simon, and M. Engeli. Spatial Modelling with Geometric Constraints. In *Practical Application of Constraint Technology*, pages 307–320, 1997.

[SOMS97] B. Seybold, G. Ogan, F. Metzger, and K. Simon. Interactive Aspects of Constrained-Based Assembly Modeling. In *the CP97 Workshop on The Theory and Practice of Dynamic Constraint Satisfaction*, 1997.

[ZGV92] B. Zalik, N. Guid, and A. Vesel. FLEXI: An experimental constraint-based Modeling System. In T. L. Kunii, editor, *Visual Computing: integrating computer graphics with computer vision.* Springer Verlag Tokio, 1992.