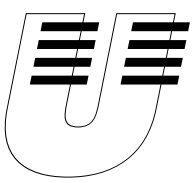


UNIVERSITÄT
KOBLENZ · LANDAU

Objektorientierte Implementierung einer Constraint basierten geometrischen Modellierung

Manfred Rosendahl

12/2004

Fachberichte
INFORMATIK

Universität Koblenz-Landau
Institut für Informatik, Universitätsstr. 1, D-56070 Koblenz

E-mail: researchreports@uni-koblenz.de,
WWW: http://www.uni-koblenz.de/fb4/

Objektorientierte Implementierung einer Constraint basierten geometrischen Modellierung

Manfred Rosendahl
Universität Koblenz
Institut für Informatik

Einleitung

In CAD Systemen möchte man nicht nur die geometrischen Objekte definieren, sondern man möchte auch funktionale Abhängigkeiten zwischen den geometrischen Objekten bzw. zwischen sonstigen Variablen, z.B. Materialkonstanten oder Belastungen, und den geometrischen Objekten definieren. Dies geschieht mit Hilfe von Constraints. Diese können z.B. festlegen, dass der Abstand von 2 Punkten ein veränderliches Maß haben soll, oder das 2 Linien gleichlang oder parallel sein sollen. Die Techniken zur Lösung dieser Geometrischen Constraint Systeme können in 3 Klassen eingeteilt werden:

- Gleichungsbasierte Methoden.
  Hier werden die Constraints durch i.A. nichtlineare Gleichungen zwischen den Variablen dargestellt. Für das nichtlineare Gleichungssystem wird entweder mit Hilfe des Newton -Raphson Verfahrens eine iterative Lösung gesucht [Light& Gossard, 1982] oder es wird eine algebraische Methode mit Hilfe von Gröbner Basen benutzt [Hoffmann, 1989] [Kondo, 1992].
- Konstruktive Techniken.
  Hier arbeitet man entweder mit Regelbasierten [Verroust, Schonek, Roller, 1992] oder Graphbasierten Techniken.
  Regelbasierte Techniken führen zur Konstruktion von Control Sets aus Längen und Winkeln. Diese Controls Sets definieren Dreiecke oder Vierecke, die dann zur Gesamtlösung zusammengefasst werden.
  Graphbasierte Techniken modellieren die Constraints als Hyperkanten zwischen der geometrischen oder sonstigen Variablen. Statt mit einem Hypergraphen, kann man auch mit einem bipartiten Graphen arbeiten, beidem jeweils Constraints und Variablen durch zunächst ungerichtete Kanten verbunden sind. Entsprechend der Anzahl der Constraints können nun aus einer gegebenen Teilmenge der Variablen, die übrigen berechnet werden.
  Bei einem rein parametrischen System liegt die Teilmenge der freien Variablen (Parameter) fest, und die übrigen können dann sequentiell berechnet werden. Daher erfolgt auch die Konstruktion des Modells in sequentiellen Schritten. Ein Ansatz in [Du, Rosendahl, Berling, 1993] fügt nachträglich, wo nötig, in einem solchen System noch zusätzliche Constraints ein, die dann Zyklen bewirken. So können auch nicht sequentiell konstruierbare Modelle definiert werden. Im Gegensatz zur Gleichungsbasierten Methode muss man hier nur ein Gleichungssystem mit minimaler Ordnung, häufig sogar nur eine Gleichung lösen.
  Im allgemeinen Fall will man jedoch nach Erstellung des Modells frei wählen können, welche Variablen vorgegeben und welche berechnet werden. Z.B. soll nicht nur ein Punkt als Schnittpunkt von 2 Linien berechnet werden, sondern bei einer Verschiebung des Schnittpunktes sollen die Linien sich adäquat bewegen. Um aus dem Modell und den festgelegten Variablen eine Berechnungsfolge zu konstruieren wird eine Analyse der Freiheitsgrade benutzt. Ziel ist es aus dem ungerichteten

Constraint- Graphen einen gerichteten zu erzeugen. Ist eine sequentielle Berechnung möglich, soll auch ein azyklischer Graph entstehen. In manchen Fällen, wo zyklische Abhängigkeiten bestehen, kann jedoch nur ein Graph mit Zyklen erzeugt werden. Sind zu viele Variablen festgelegt, d.h. das System ist überbestimmt ist keine Orientierung möglich. Ansätze in dieser Richtung liefern [Berling, 1996], [Fudos, Hoffmann, 1993], [Hsu, Brüderlin, 1997].

Die vorliegende Arbeit führt den Ansatz von [Berling, 1996] weiter. Es wird jedoch eine reichere Auswahl von Constraintklassen betrachtet. Ferner wird bei der Analyse der Freiheitsgrade auch berücksichtigt, dass manche Constraints nicht unabhängig von einander sind. So hat ein Punkt im 2D zwar 2 Freiheitsgrade, die jedoch nicht durch 2 Constraints abgedeckt werden können, die bei nur die X oder Y Koordinate beeinflussen. Ferner dürfen die durch die Constraints gegebenen Ortskurven nicht Kreise mit dem gleichen Mittelpunkt oder parallele Linien sein. Des Weiteren wurde Wert darauf gelegt, dass auch stark unterbestimmte Systeme ein sinnvolles Verhalten zeigen. Daher haben Constraint nicht nur auf die direkt berührten Punkten Auswirkungen, sondern sie werden auch auf entfernter liegende Punkte propagiert.

Geometrische Modellierung

Bei der Erstellung eines geometrischen 2D-Modells hat man pro Punkt 2 Freiheitsgrade und einen Freiheitsgrad pro Skalarwert z.B. Radius eines Kreises. Diese Freiheitsgrade werden eingeschränkt durch die zu erfüllenden Constraints. Die Constraints z.B. der Abstand zweier Punkte nehmen jeweils einen Freiheitsgrad weg. Eine Konstruktion kann also maximal so viele Constraints erfüllen, wie Freiheitsgrade vorhanden sind.
Beispiel: Eine Linie gegeben durch 2 Endpunkte.
Freiheitsgrade: 4 (X1,Y1,X2,Y2)
Gibt man die Länge vor hat man noch 3 Freiheitsgrade. Fordert man, dass die Linie waagrecht, so bleiben noch 2 Freiheitsgrade.
Man kann jetzt noch die Koordinaten eines Punktes festlegen oder die X-Koordinate des einen und die Y-Koordinate des anderen Punktes. Man kann jedoch nicht die X oder Y Koordinaten beider Punkte festlegen, da diese durch die vorherigen Constraints bereits von einander abhängig sind. Man muss daher bei dem Verfolgen der Constraints nach halten, auf welche Koordinaten der Constraint Einfluss hat. Der Constraint Waagrecht beeinflusst nur die Y-Koordinaten.
Ferner muss man berücksichtigen, dass manche Constraints die Lage der Punkte untereinander beeinflussen andere die Lage und Ausrichtung der Gesamtkonstruktion.
So beeinflusst der Abstand zweier Punkte oder der Constraint, der festlegt, dass 3 Punkte einen rechten Winkel bilden, nur die Lage der Punkte zu einander. Für diese Lage zu einander, d.h. die kongruente Form, gibt es bei n Punkten jedoch nur 2n-3 Freiheitsgrade. Daher kann man bei einem Dreieck, bei 3 Seitenlängen, nicht noch einen Winkel bestimmen.

Die Punkte, Skalarvariablen und Constraint bilden die Knoten eines bipartiten Graphen. Die eine Knotenklasse bilden die Punkte und die skalaren Variablen (diese Klasse wird im Folgenden Variablen genannt), die andere die Constraints. Die Kanten stellen die Beziehungen zwischen den Constraints und den Variablen dar.

Beispiel: 2 Punkte mit Abstand und Waagrechter Ausrichtung.

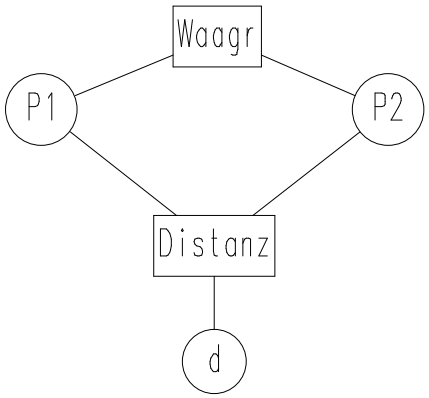


Abbildung 1

Die Variablen werden später im gerichteten Graphen mit maximal so vielen einlaufenden Kanten verbunden, wie sie Freiheitsgrade haben. Ist der Wert der Variablen festgelegt, werden die adjazenten Kanten später alle als auslaufend orientiert. Wird eine Variable für mehrere Constraints genutzt, kann nur eine Kante einlaufend sein.

Die Constraintknoten haben eine Kantenverbindung zu jeder Variablen, die durch sie beeinflusst wird. In der Literatur wird i.A. mit Constraints gearbeitet, die eine feste Anzahl von Freiheitsgraden binden. Ein Constraint, das den Abstand von 2 Punkten bestimmt, wird mit 2 Punkten und einem Skalarwert verknüpft. Entweder bestimmen die beiden Punkte den Abstand oder der Abstand und ein Punkt legt für den anderen Punkt eine Ortskurve fest. Man sagt dann der Abstand- Constraint konsumiert einen Freiheitsgrad. In der hier betrachteten Modellierung haben die Constraints jedoch einen vorgegebenen Freiheitsgrad. Der Constraint Distanz hat 2 Freiheitsgrade, d.h. maximal 2 der adjazenten Kanten können einlaufend orientiert sein. Dies könnte wieder sein:

- Die beiden Punkte: Dann wird der Abstand d festgelegt.
- Ein Punkt und der Abstand d: Dann wird für den anderen Punkt eine Ortskurve, hier Kreis um den ersten Punkt mit dem Radius d.

Ein Constraint konsumiert so viele Freiheitsgrade der Variablen, wie er auslaufende Kanten hat, d.h. mindestens Anzahl der adjazenten Kanten – Freiheitsgrad.
Hier können einige Constraints eine variable Anzahl von adjazenten Kanten haben. Der Constraint Waagrecht z.B. bestimmt, dass eine Reihe von Punkten die gleiche Y-Koordinate haben soll. Es kann eine Kante einlaufend orientiert werden. Der adjazente Punkt bestimmt die Y-Koordinate der anderen Punkte. Ähnliches gilt für den Constraint Kreis. Neben Mittelpunkt und Radius können noch beliebig viele auf den Kreis liegende Punkte, adjazent sein. Nun kann entweder durch 3 Punkte auf dem Kreis Mittelpunkt und Radius bestimmt werden, oder Mittelpunkt und Radius bestimmen jeweils eine Ortskurve für die Punkte auf dem Kreis. Natürlich sind auch alle anderen Kombinationen möglich, wie z.B. 2 Punkte auf dem Kreis und der Radius bestimmt den Mittelpunkt. Um alle Fälle für die verschiedenen Constraints abzudecken, hat jede Constraintklasse eine Methode compute, die für alle adjazenten Variablen, mit auslaufender Kante den Wert bzw. eine Ortskurve bestimmt.

Bei der Auswertung wird entsprechend der vorgegeben Variablen aus dem ungerichteten Graphen ein gerichteter Graph erzeugt. Dieser wird dann, falls möglich, in der Reihenfolge der topologischen Sortierung ausgewertet. Es ist jedoch nicht immer möglich, den Graphen kreisfrei zu orientieren. Dann werden in dem Graphen die starken Komponenten ermittelt. Das ist eine Aufteilung der Knotenmenge in Teilmengen so, dass die Zyklen alle innerhalb der starken Komponenten liegen. Die starken Komponenten mit mehr als einem Knoten werden im Allgemeinen durch eine Iteration berechnet. Dies führt jedoch nicht immer zu einem Ergebnis. Daher werden später Verfahren vorgestellt, die für einige häufig vorkommende Fälle eine direkte Lösung erlauben. Die Reihenfolge der Berechnung der starken Komponenten kann nach der topologischen Sortierung erfolgen.

Beschreibung der Constraints

	Freiheits-grad	Adjazente Knoten	Beschreibung	Bestimmte Koordinate
Twaagrecht	1	Punkte, die waagrecht ausgerichtet werden	Ein Punkt bestimmt die Y-Koordinate der anderen Punkte	Y
Tsenkrecht	1	Punkte, die senkrecht ausgerichtet werden	Ein Punkt bestimmt die X-Koordinate der anderen Punkte	X
Tdistanz	2	2 Punkte und ein Skalar	2 Punkte bestimmen ein Maß oder das Maß und ein Punkt bestimmen den anderen Punkt	XY
Tdistanzx	2	2 Punkte und ein Skalar	2 Punkte bestimmen ein Maß oder das Maß und ein Punkt bestimmen den anderen Punkt	X
Tdistanzy	2	2 Punkte und ein Skalar	2 Punkte bestimmen ein Maß oder das Maß und ein Punkt bestimmen den anderen Punkt	Y
Twinkel	2	2 oder mehr Punkte und ein Skalar	2 Punkte bestimmen einen Winkel oder der Winkel und ein Punkt bestimmen die anderen Punkte	XY
Twinkel3P	3	3 Punkte und ein Skalar	3 Punkte bestimmen einen von ihnen aufgespannten Winkel oder der Winkel und 2 Punkte bestimmen den 3. Punkt	XY
Trechtwinklig	2	3 Punkte	Wie Winkel3P, jedoch ist der Winkel auf 90° festgelegt	XY
Tkreis	3	Radius, 2 Kanten zum Mittelpunkt und weitere Punkte auf dem Kreis	Der Kreis wird durch 3 einlaufende Kanten bestimmt. Die auslaufenden Kanten	XY

			bestimmen den Radius oder Ortskurven für den Mittelpunkt bzw. die Punkte auf dem Kreis.	
Tlinie	2	2 oder mehr Punkte, die auf einer Geraden liegen	2 einlaufende Kanten bestimmen die Gerade auf der alle Punkte liegen. Die auslaufenden Kanten bestimmen Ortskurven für die anderen Punkte.	XY
Tgleichmx	2	2 oder mehr Punkte, die horizontal gleichmäßig verteilt werden	2 einlaufende Kanten bestimmen die X-Koordinaten von 2 Punkten. Die restlichen Punkte werden horizontal gleichverteilt	X
Tgleichmy	2	2 oder mehr Punkte, die vertikal gleichmäßig verteilt werden	2 einlaufende Kanten bestimmen die Y-Koordinaten von 2 Punkten. Die restlichen Punkte werden gleichverteilt	Y
Tdistanzlp	3	Skalar (Abstand) und 2 Punkte, die eine Gerade definieren und ein Punkt, dessen Abstand bestimmt wird	3 Punkte bestimmen den Abstand oder Abstand und 2 Punkte bestimmen den dritten Punkt	XY
Toperator	2	2 Skalare als Operanden und ein Skalar als Ergebnis	Die Operanden bestimmen das Ergebnis oder ein Oparand und das Ergebnis bestimmen den anderen Operanden	XY
Tbogen	3	Radius, 2 Kanten zum Mittelpunkt, Anfangspunkt, Endpunkt des Bogens und weitere Punkte auf dem Bogen	Der Kreis wird durch 3 einlaufende Kanten bestimmt. Die auslaufenden Kanten bestimmen den Radius oder Ortskurven für den Mittelpunkt bzw. die Punkte auf dem Bogen.	XY

Andere Constraints lassen sich aus diesen zusammensetzen. Sollen 2 Linien, die gleiche Länge haben, werden 2 Distanz Constraints definiert, deren Abstand identifiziert wird.

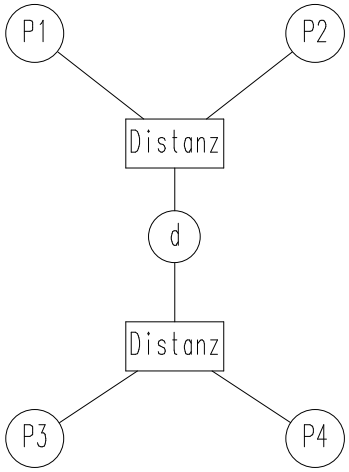


Abbildung 2

Analog kann definiert werden, wenn 2 Linien parallel sein sollen

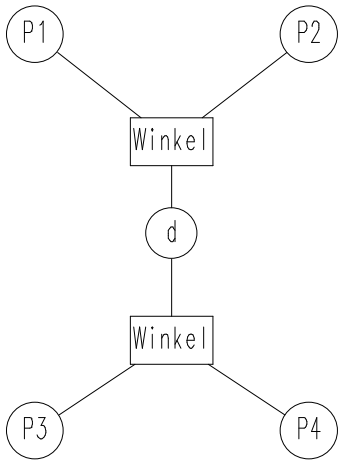


Abbildung 3

Ein Punkt kann durch zwei Linien Constraints als Schnittpunkt von 2 Linien definiert werden.

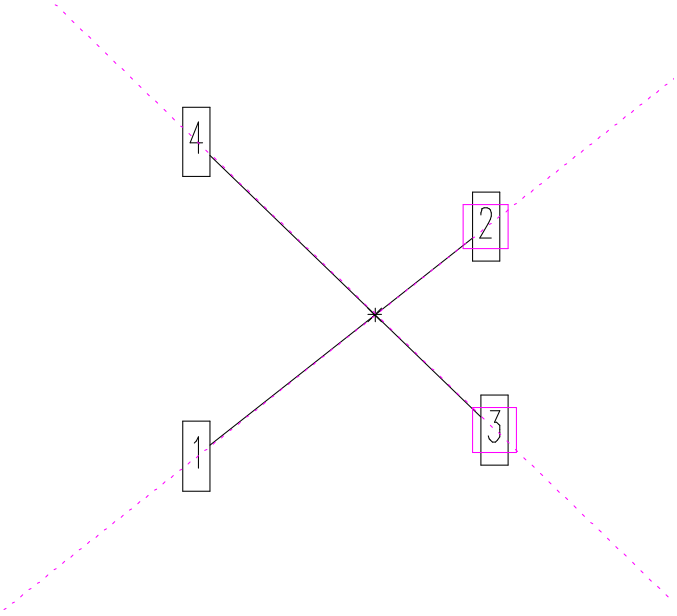


Abbildung 3a

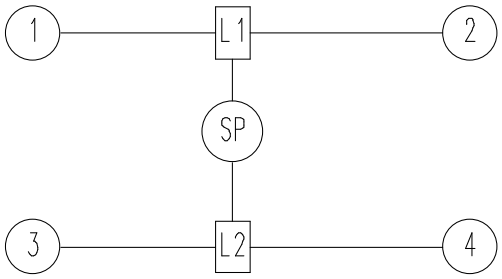


Abbildung 3b

Der Schnittpunkt muss auf den Linien der Punkte 1-2 und der Punkte 3-4 sein. Wird der Schnittpunkt verschoben, so ändern sich die Punkte 1 und 4, da 2 und 3 fixiert sind. Die Orientierung des Graphen ist dann:

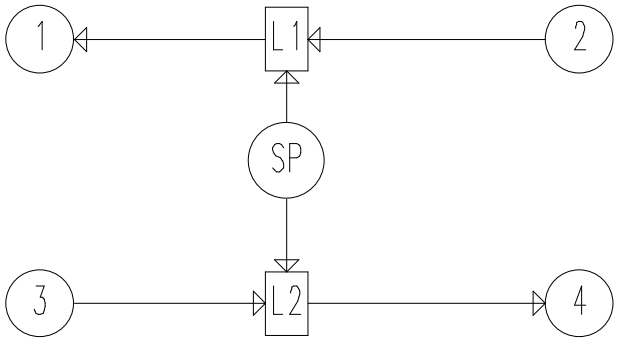
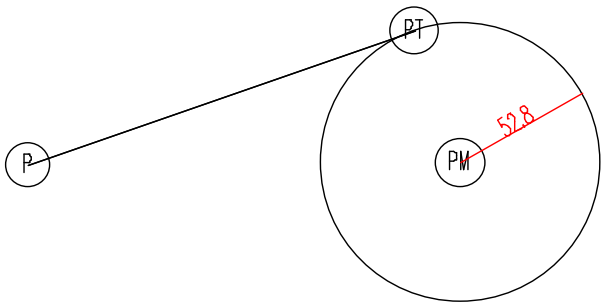


Abbildung 3c

Ändern sich die Punkte 1-4 ändert sich analog der Schnittpunkt.

Auch eine Linie, die von einem Punkt aus tangential an einen Kreis geht, kann über Kreis und Rechtwinklig definiert werden.



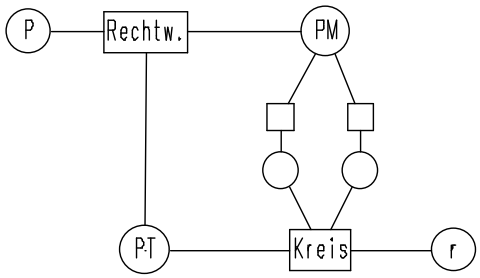


Abbildung 4

Da der Mittelpunkt 2 Freiheitsgrade des Kreises bestimmt und es in der Graphdarstellung nur jeweils eine Kante zwischen 2 Knoten geben soll, müssen zwischen Mittelpunkt und Kreis Zwischenknoten eingefügt werden. Damit der Graph bipartit bleibt, muss ein Constraint-Knoten und ein Variablen- Knoten eingefügt werden.

Sollen 2 Kreise sich tangential berühren, so kann dies über eine Abstandsbeziehung der Mittelpunkte definiert werden. Der Abstand der Mittelpunkt muss die Summe bzw. die Differenz der Radien ergeben.

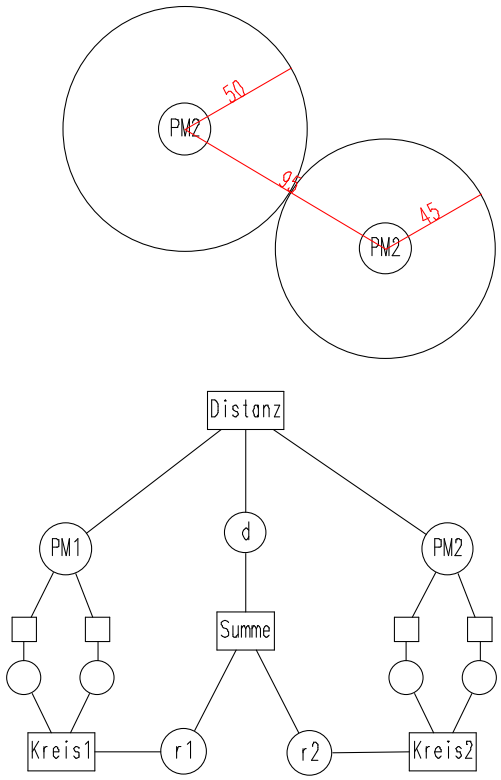
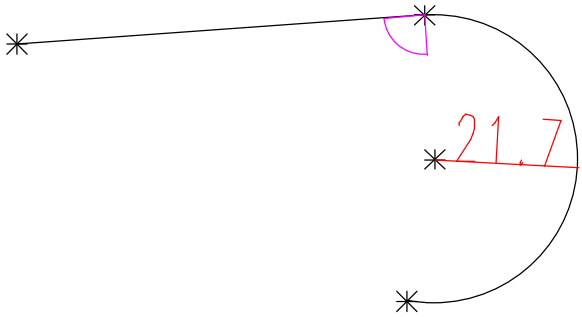


Abbildung 5a

Durch entsprechende Orientierung des Constraintgraphen können sowohl bei einer Änderung der Radien die neuen Mittelpunkte berechnet werden, als auch bei einer Änderung der Mittelpunkte neue Radien ermittelt werden. Die Konstruktion hat insgesamt 7 Freiheitsgrade (2 Punkte, 2 Radien und ein Abstand). Die Kreise binden keinen Freiheitsgrad, da sie 3 Freiheitsgrade und auch nur 3 adjazente Variablen haben. Die Distanz hat 2 Freiheitsgrade und 3 adjazente Kanten, bindet also einen Freiheitsgrad. Für die Summe gilt das gleiche. Insgesamt bleiben also 5 Freiheitsgrade übrig, z.B. Koordinate eines Mittelpunktes, 2 Radien und Richtung des anderen Mittelpunktes oder auch 2 Mittelpunkte und ein Radius.

Soll ein Bogen tangential an eine Linie anschließen, so muss der Endpunkt der Linie, der Anfangspunkt des Bogens sein, ferner muss er tangential sein, d.h. die beiden Linienpunkte und der Kreismittelpunkt bilden einen rechten Winkel.



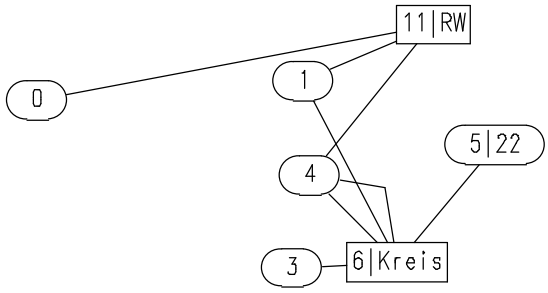
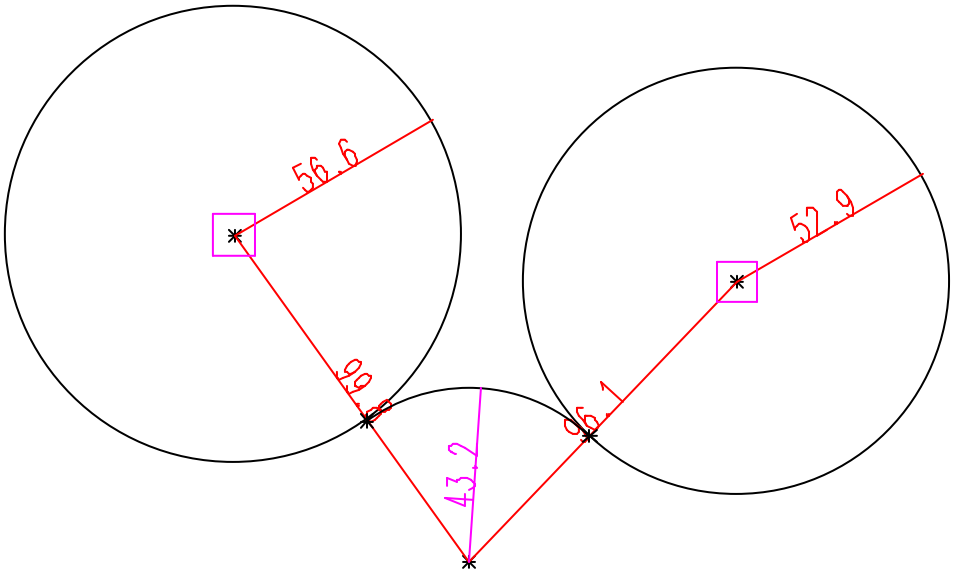


Abbildung 5b



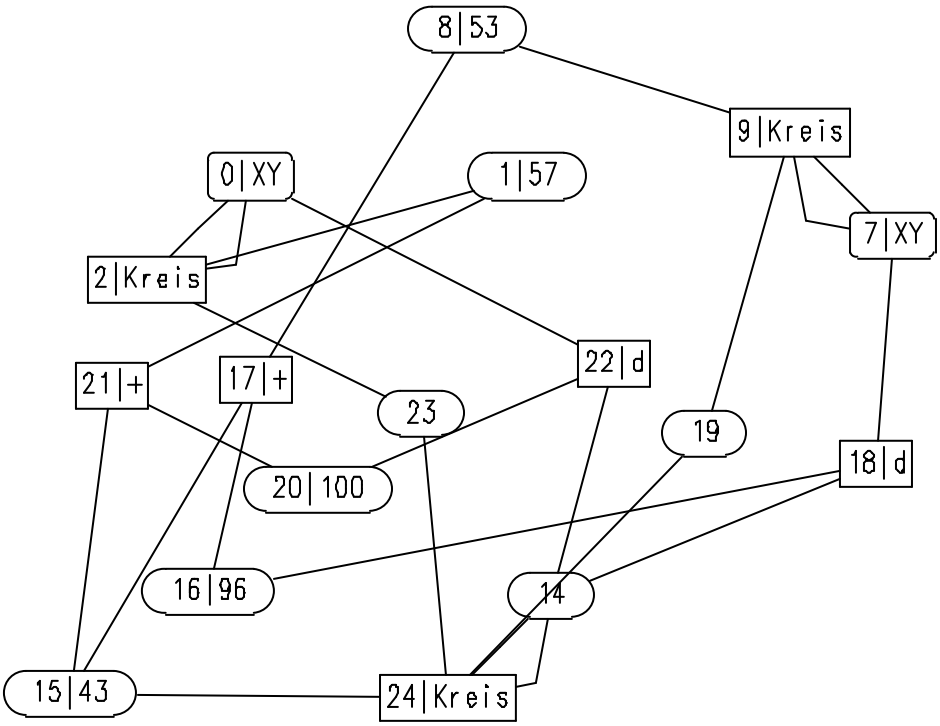


Abbildung 5b

Der Bogen sei tangential an den beiden Kreisen. Damit ist der Abstand der Bogenmittelpunktes zu den Kreismittelpunkten jeweils die Summe des Bogenradius und des Kreisradius.(Constraints 17 und 21).

Im obigen Beispiel sind der Bogenradius und die Kreismittelpunkte vorgegeben. Wird der Bogenmittelpunkt verschoben, passen sich die Kreisradien jeweils an. Der orientierte Graph ist sogar zyklenfrei.

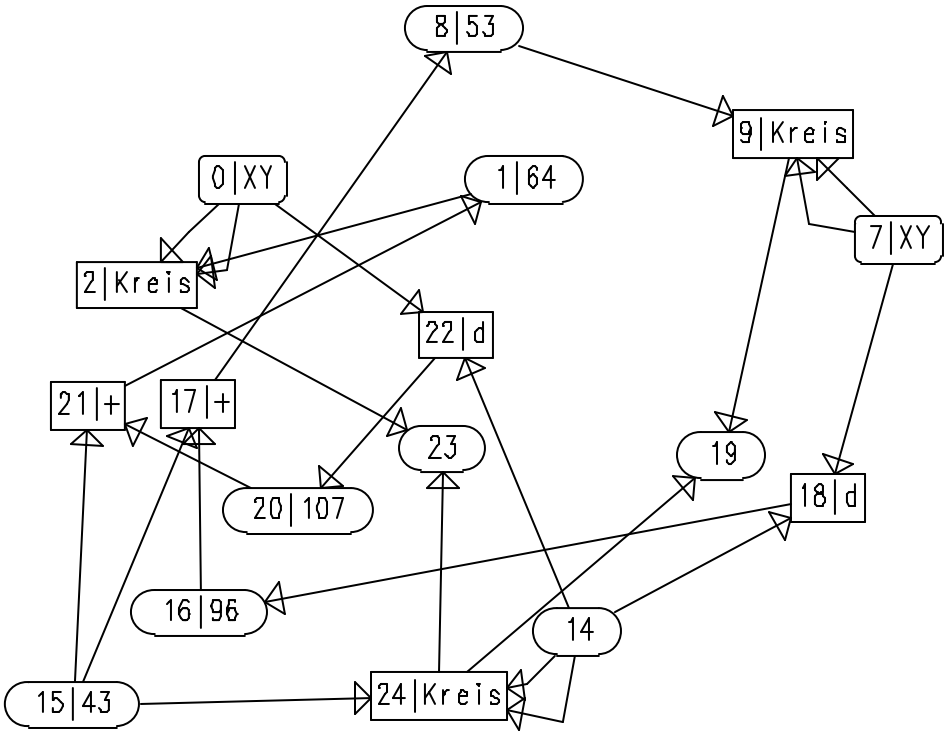


Abbildung 5c

Orientierung des Constraintgraphen

Durch einlaufende Kanten wird der Freiheitsgrad der Variablen reduziert. Ist der Freiheitsgrad auf 0 reduziert, müssen alle weiteren Kanten auslaufend orientiert werden. Wie oben gezeigt beeinflussen manche Constraint jedoch nur die X bzw. Y Koordinate eines Punktes. Daher kann der Freiheitsgrad nicht einfach runtergezählt werden, sondern es muss folgende Transitionsmatrix benutzt werden:

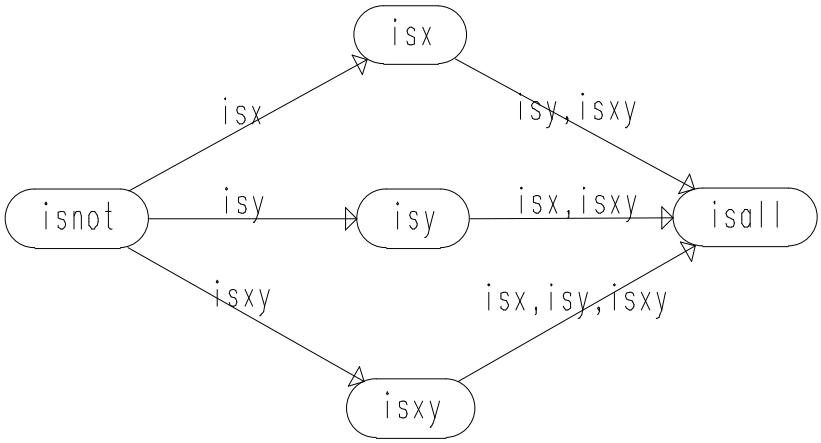


Abbildung 6

Ein Punkt hat zunächst 2 Freiheitsgrade, was dem Zustand `isnot` entspricht. Wird durch ein Constraint die X-Koordinate bestimmt, geht er in den Zustand `isx` über. Nun kann er nur noch ein Constraint erfüllen, was auch die Y-Koordinate verändert. Dies kann entweder ein Constraint sein, das nur die Y-Koordinate bestimmt (`isy`) oder ein Constraint, was eine Ortskurve festlegt (`isxy`). Insgesamt kann ein Punkt also folgende Constraintpaare erfüllen:

- `Isx, isy`
- `Isx, isxy`
- `Isy, isxy`
- `Isxy, isxy`

Auch die Funktion, die für einen Punkt, die noch vorhandenen Freiheitsgrade berechnet, hat als Parameter was durch zusätzlich fixiert werden soll:

```
function tnod.sdf(afix:tfix):integer;
begin
result:=0;
case fix2 of
 isnot:result:=2;
 isx:if afix=isx then result:=0 else result:=1;
 isy:if afix=isy then result:=0 else result:=1;
 isxy:result:=1;
end;
end;
```

`fix2` gibt den derzeitigen Zustand des Punktes an. Ist z.B. die X Koordinate bereits fixiert `(isx)`, so hat er für ein Constraint, das auch nur die X Koordinate fixiert, keinen Freiheitsgrad mehr, für andere Constraints `(isy,isxy)`, die auch die Y Koordinate beeinflussen, jedoch noch einen Freiheitsgrad.

Ist es möglich, einen Constraintgraphen zyklenfrei zu orientieren, kann diese Orientierung auch gefunden werden. Eine möglich zyklenfreie Orientierung, kann jedoch nicht immer berechnet werden.
Beispiel: Gegeben seien 3 Punkte, die durch 2 Linien mit je einem Abstandconstraint verbunden sind.

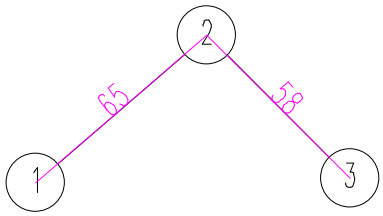


Abbildung 7

Wird der Punkt 3 soweit verschoben, dass sein Abstand vom Punkt 1 größer als die Summe der Abstände ist, wird man für den Punkt 2 keine Lösung mehr finden. Falls der Punkt 1 nicht festgelegt ist muss also die Verschiebung von 3 auch nach 1 propagiert werden, damit 1 entsprechend nachgezogen wird. Der orientierte Constraintgraph hat dann folgende Form:

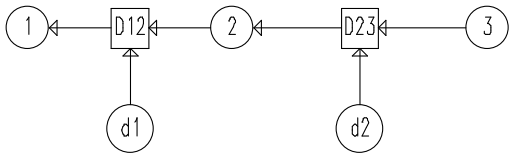


Abbildung 8

Für den Punkt 2 gibt es dann nur die Ortskurve Kreis um 3 mit Radius d2. Von der bisherigen Position von 2 wird dann ein Lot auf diesen Kreis gefällt. Das gleiche geschieht bei Punkt 1. Verschiebt man Punkt 3 so werden die Punkt 2 und 1 nachgezogen, wobei sich die Anordnung immer mehr einer geraden Linie nähert.

Algorithmus zur Orientierung des Constraintgraphen

Bevor versucht wird den ungerichteten Graphen zu orientieren, wird zunächst getestet, ob die Konstruktion nicht überbestimmt ist.

Für die einzelnen Constraints muss untersucht werden, ob sie die Form, die Lage oder die Ausrichtung bestimmen.

Die Constraints tdistanz, twinkel3p, tdistanzlp verringern jeweils die Formfreiheitsgrade um eins, wenn ihr Wert vorgegeben ist, bzw. wenn der gleiche Wert schon in einem anderen Constraint benutzt wurde.
Trechtwinklig verringert der Formfreiheitsgrad jeweils um ein, tlinie um die Anzahl der Punkte-2.
Die Constraints tkreis und tbogen verringern die die Formfreiheitsgrade für jeden Punkt, der auf dem Kreis liegen muss.
Bei den Constraints twaagrecht, tsenkrecht, twinkel bestimmen der erste Constraint und die ersten beiden Punkte die Ausrichtung, alle weiteren Punkte und alle weiteren Constraints verringern die Formfreiheitsgrade.

Der erste fixierte Punkt bestimmt die Lage der Konstruktion. Weitere Fixierungen reduzieren die Formfreiheitsgrade. Ferner können Fixierungen von Punkten und Constraints, z.B. über Abstände zu Abhängigkeiten führen, was die Konstruktion ebenfall überbestimmt macht.

Für die Abstände zwischen Punkten gilt:

```
dx(P1-P2) , dx(P2-P3) -> dx(P1,P3)
dy(P1-P2) , dy(P2-P3) -> dy(P1,P3)
dx(P1-P2) , dy(P1,P2) -> d(P1,P2)
```

Werden Punkte fixiert gilt:

```
Fixx(P1) , Fixx(P2) -> dx(P1,P2)
Fixy(P1) , Fixy(P2) -> dy(P1,P2)
Fixx(P1),Fixy(P1),Fixx(P2),Fixy(P2)-> d(P1,P2)
```

Diese Bedingungen werden durch Matrizen überprüft

```
Md[n,m]=true  <-> d(Pn,Pm)  fixiert
Mdx[n,m]=true <-> dx(Pn,Pm) fixiert
Mdy[n,m]=true <-> dy(Pn,Pm) fixiert

Fixx[n]=true  <-> x Koordinate von Pn fixiert
Fixy[n]=true  <-> y Koordinate von Pn fixiert
```

Für jeden Abstand- Constraint und jede Punkt Fixierung wird die entsprechende Matrix gesetzt.

Für **Mdx** und **Mdy** wird jeweils die Hülle gebildet

```
Md := Md + Mdx*Mdy
Fixx[n] and Fixx[m] -> Mdx[n,m]=true
Fixy[n] and Fixy[m] -> Mdy[n,m]=true
Fixx[n] and Fixy[n] and Fixx[m] and Fixy[m] ->
Md[n,m]=true
```

Ist die entsprechende Position in der Matrix schon gesetzt, ist die Konstruktion überbestimmt

Schließlich muss noch überprüft werden, ob n Constraints von Winkel zwischen 3 Punkten (trechtwinklig oder twinkel3p) ein geschlossenes Polygon mit n Punkten bilden. Dann ist die Konstruktion überbestimmt, da in einem geschlossenen Polygon von n Punkten nur n-1 Winkel unabhängig sind.
Ergibt die Analyse der Freiheitsgrade nicht, dass die Konstruktion überbestimmt ist, wird eine möglichst zyklenfreie Orientierung des Graphen gesucht.

Im ersten Schritt werden alle vorgegeben Fixierungen erfüllt. Dies können sein, Fixierungen der Werte bzw. Koordinaten durch sich durch vorgegebene Maße oder Punkte ergeben oder Fixierungen, die sich aus der Operation ergeben. Soll ein Punkt an eine bestimmte X, Y Koordinate verschoben werden ist er fixiert, d.h. im Zustand `isall`.

Für die Orientierung werden folgende Funktionen benutzt:
`v.sdf(c.fix)` = noch vorhandener Freiheitsgrad in Knoten v für das Constraint c
`c.sdf` = noch vorhandener Freiheitsgrad in Constraint c, das ist der Freiheitsgrad – Anzahl der einlaufenden Kanten.

Bevor jeweils neue Kanten orientiert werden, wird geprüft, welche Kanten zwangsweise orientiert werden müssen:
Für jede Variable `v` und jeden adjazenten Constraint `c`, mit noch ungerichteter Kante:

```
if (v.sdf(c.fix)=0)then begin
 if (c.sdf>0)then begin
  kantenrichtung(v,c);change:=true;
 end else begin
  result:=false;exit; //Kante kann nicht orientiert werden
 end;
end;
```

Für jeden Constraint c und jede adjazente Variable v, mit noch ungerichteter Kante:

```
if (c.sdf=0)and(v.sdf(c.fix)>0) then begin
 kantenrichtung(c,v);change:=true;
end else begin
  result:=false;exit;// kante kann nicht orientiert werden
 end;
end;
```

Für alle Constraints c, wo die Anzahl der noch ungerichteten Kanten = der Anzahl der noch vorhandenen Freiheitsgrade ist;

```
j:=c.fnedges[isnone]; // Anzahl der noch ungerichteten Kanten
if (j>0)and(j<=c.sdf) then begin
 for j:=0 to c.count-1 do if c.ori[j]=isnone then begin
  v:=c.vars[j];
  kantenrichtung(v,c);change:=true;
 end;
end;
```

Von den Variablen, d.h. den Punkten oder Skalaren, die mit der Operation geändert werden sollen, ausgehend, wird eine Breitensuche im Constraintgraphen durchgeführt. Entsprechend der Reihenfolge, in der die Variablen -Knoten bei der Breitensuche durchlaufen wurden, wird eine Reihenfolge festgelegt.

In dieser Reihenfolge werden die noch ungerichteten Kanten der Variablen auslaufend orientiert. Nach jeder einzelnen Orientierung werden wieder die Zwangsorientierungen geprüft.

Hiermit werden die Constraints von den zu verändernden Variablen ausgehend propagiert.

Beispiel:

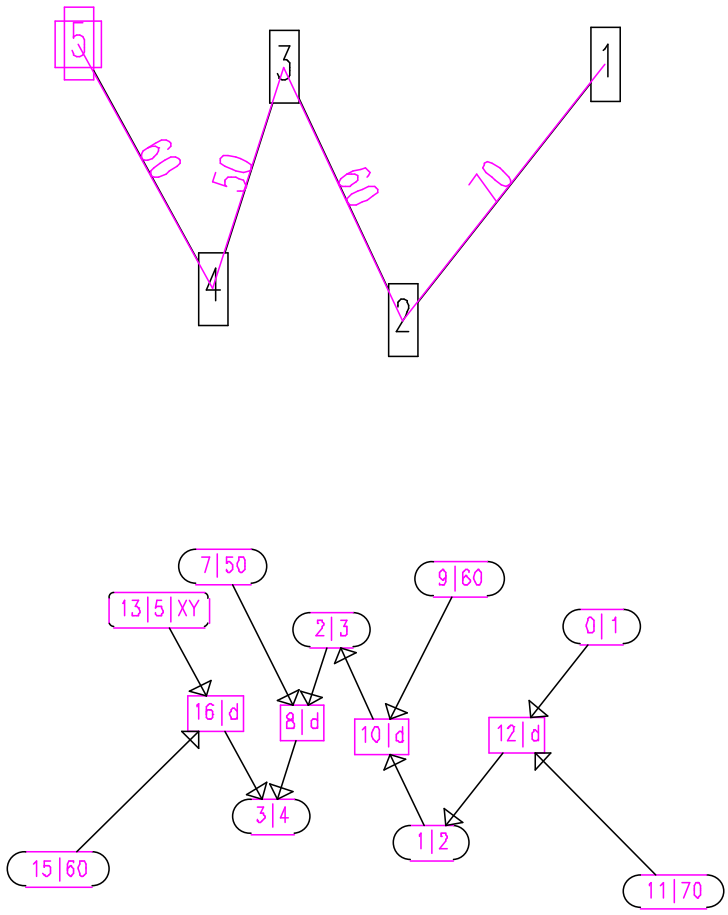


Abbildung 9

Der Knoten 1 wird verschoben. Die Änderung wird bis zum Knoten 4 propagiert, der seine 2. Ortskurve über den Abstand zum Knoten 5 erhält, dessen Position fixiert ist.

Kann der Graph mit diesem Verfahren zyklenfrei orientiert werden, wird diese Orientierung hiernach auch benutzt. Bleiben nach dem obigen Algorithmus noch ungerichtete Kanten übrig, oder enthält die Orientierung Zyklen, wird versucht die Lösung mit der minimalen Anzahl von Zyklen zu finden:

Dazu gibt es 2 Möglichkeiten:

1. Methode: Matching erweitern

Kann eine Kante nicht orientiert werden, so ist an beiden adjazenten Knoten kein Freiheitsgrad mehr vorhanden. Um die Kante zu orientieren, muss also an einem adjazenten Knoten eine einlaufende Kante in eine auslaufende verwandelt werden. Jedoch muss dann der dritte Knoten, der mit dieser Kante adjazent ist, noch einen Freiheitsgrad haben. Ist dies nicht der Fall, muss man an diesem Knoten auch wieder eine Kantenrichtung ändern. Dies wird solange wiederholt, bis man an einen Knoten mit einem noch vorhandenen Freiheitsgrad stößt, oder man gelangt auf dem Weg zu einem Knoten der schon auf dem Weg ist. Hat man einen Knoten mit Freiheitsgrad gefunden, werden alle Kanten auf dem Weg umorientiert, andernfalls wird die Suche abgebrochen. Diese Suche, die auch als Bestimmung eines maximalen Matchings in bipartiten Graphen (McHugh, 1990, Seite 200ff) bekannt ist, wird hier durch die rekursive Funktion `suchen` realisiert.

```
function suchen(a,a1:tany;l:tanylist):boolean;
// Es wird von a1 kommend der Weg über a hinaus verlängert
var v:tvar;
 c:tconstr;
 i:integer;
label 1,99;
begin
result:=false;
if a.mark2 then exit; //schon besucht
a.mark2:=true;
if a is tvar then begin
 v:=a as tvar;c:=a1 as tconstr;
 if v.sdf(c.fix)>0 then goto 99 // a hat noch Freiheitsgrad
 else begin
  1:
  for i:=0 to a.count-1 do if a.ori[i]=isin then begin
   if suchen(a.kanten[i],a,l)then goto 99;
  end;
 end;
 exit;
end else begin
 c:=a as tconstr;
 if c.sdf>0 then goto 99 else goto 1;
end;
99:result:=true;l.add(a);
// l enthält den Weg von hinten nach vorne
end;
```

War `suchen` mit einem adjazenten Knoten der noch ungerichteten Kante erfolgreich, werden alle Kanten auf dem Weg umorientiert und dann kann die Kante bei diesem Knoten einlaufend orientiert werden.

Kann bei den beiden adjazenten Knoten, der ungerichteten Kante, kein Weg zu einem Knoten mit noch vorhandenen Freiheitsgrad gefunden werden, so ist das Constraint System überdefiniert und es existiert keine Lösung.

Hierbei wird eine Lösung gefunden, die nicht optimal sein muss, d.h. die evtl. nicht die wenigsten Zyklen hat.

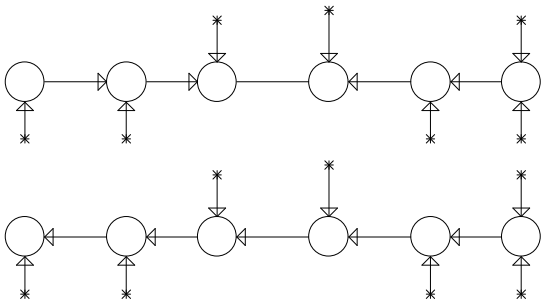


Abbildung 10

2. Methode: Brute Force:
Es werden systematisch alle möglichen Orientierungen untersucht, und die ausgewählt, die am wenigsten Knoten auf einem Zyklus hat.

Der Aufwand ist exponentionell mit der Anzahl der Wahlmöglichkeiten. Dies ist aber weniger als die Anzahl der Kanten, da nach einer Wahl mit dem obigen Algorithmus weitere Kanten orientiert werden. Zu einem werden nach einer Orientierung alle Zwangsorientierungen vorgenommen. Zusätzlich werden für alle Variablen, bei denen noch alle noch ungerichteten Kanten einlaufend orientiert werden können, diese auch als einlaufend orientiert. In der ersten Phase wird dies absichtlich nicht gemacht, da hierdurch die Propagierung in der Reihenfolge der Breitensuche durchbrochen werden kann.
Wird eine zyklenfreie Orientierung gefunden, wird aufgehört, sonst wird die Anzahl der Knoten in den Zyklen ermittelt. Es wird die Lösung ausgewählt, die die minimale Anzahl von Knoten in Zyklen hat. Wird bei der Lösungssuche der bisher beste Anzahl überschritten, wird dieser Weg nicht mehr verfolgt (Branch and bound).

Durch die vorherige Prüfung der Freiheitsgrade ist der Fall, dass keine Lösung existiert, und damit auf jeden Fall, der ganze Lösungsbaum durchlaufen wird, ausgeschlossen.

Evaluierung des Constraint- Graphen

Der nach der Orientierung erhaltene gerichtete Graph, kann nun jedoch Zyklen erhalten. Deswegen werden zunächst die starken Komponenten ermittelt. Hier werden Knoten zwischen denen ein Zyklus existiert, zu einer Komponente zusammengefasst. Hat der Graph keine Zyklen besteht jede Komponente aus genau einem Knoten.

Die Ermittlung der starken Komponenten erfolgt durch eine rekursive Tiefensuche. Findet man hierbei einen Knoten der bereits auf dem Weg liegt hat man einem Kreis gefunden und die Knoten auf dem Kreis gehören zur gleichen starken Komponente. Der Graph der starken Komponenten bildet einen DAG. Daher können die starken Komponenten in der Reihenfolge der topologischen Sortierung evaluiert werden.

Die Evaluierung einer starken Komponente mit mehr als einem Knoten, ist jedoch nicht sequentiell möglich.

Hier kann ein Iterationsverfahren benutzt werden. Hierbei muss jedoch nicht ein Gleichungssystem über alle Knoten in dem Graph der starken Komponente gelöst werden. Es wird vielmehr eine Variablenmenge V gesucht, die alle Kreise der Komponente dominiert. Dies nennt man die Feedback-Vertex- Menge F. Gesucht wird eine minimale Menge F. Dieses Problem ist NP-vollständig, es kann jedoch wieder mit rekursiver Tiefensuche eine Näherungslösung gefunden werden. Wählt der Algorithmus für F einen Constraintknoten C aus, wird er ersetzt durch den Variablenknoten V, der auch auf dem Kreis liegt, und der von C aus erreicht wird. Da der Graph bipartit ist, kann en Kreis nicht nur Constraintknoten enthalten. Ferner werden die Zwischenknoten zwischen einem Constraint der Klasse TKreis und dem Mittelpunkt des Kreises durch dem Mittelpunkt ersetzt.

Bei der Iteration werden in einer Schleife die Knoten aus F mit topologischer Sortierung berechnet. Wird dabei ein anderer Knoten aus F oder der gleiche Knoten benötigt, wird der alte Wert genommen. Dies wird solange gemacht, bis sich die Werte der Knoten in F nicht mehr ändern oder eine Maximalzahl von Durchläufen erreicht ist.

Ist die Evaluierung für den Gesamtgraphen nicht erfolgreich, wird zunächst der längste Kreis in dem gerichteten Graphen ermittelt. Auf diesem Kreis werden alle Kanten umgedreht. Die Zahl der ein- und auslaufenden Kanten bleibt dabei für jeden Knoten gleich. Auf den so geänderten Graphen wird dann noch mal das gleiche Verfahren angewandt.

Beispiel:

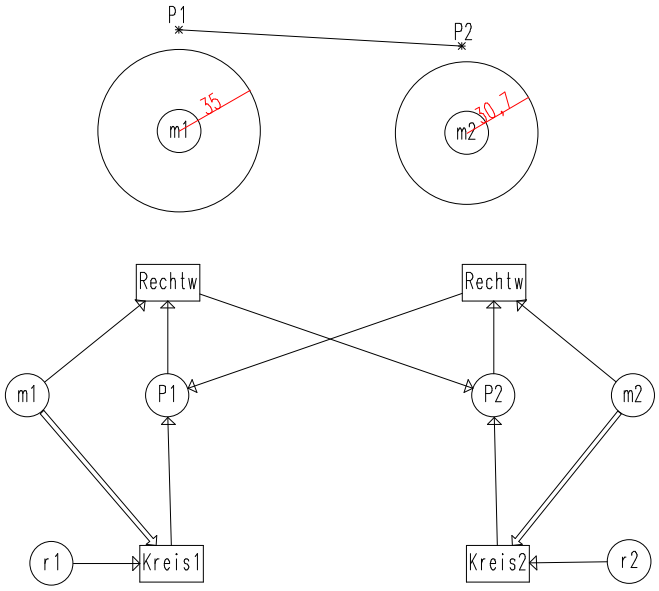


Abbildung 11

Soll die Linie tangential an die beiden Kreise gelegt werden, ist nur eine Lösung mit Kreisen im gerichteten Graphen möglich. Die starke Komponente besteht aus: P1, P2 und den beiden Rechtw Constraint. Bei der Evaluation werden zunächst P1 und P2 auf die Kreise gelegt. Dann werden abwechselnd jeweils die Rechtw Constraint erfüllt. Nach wenigen Iterationen bleibt die Position von P1 und P2 stabil.

Bei der Evaluierung des kreisfreien Graphen der starken Komponenten wird für die Komponenten, die nur aus einem Knoten bestehen, jeweils nachdem alle Knoten, von denen Kanten einlaufen berechnet sind, der Knoten berechnet. Dazu wird für den Knoten die Methode `compute` aufgerufen.
Bei den Variablenknoten (tv und tnod) wird wie folgt verfahren:

`tv.compute:` keine Aktion. Der Wert ist durch den mit einer einlaufenden Kante verbundenen Constraint-Knoten schon berechnet.
`Tnod.compute:` Durch die mit einer einlaufenden Kante verbundenen Constraint-Knoten sind für den Punktknoten maximal 2 Ortskurven bestimmt. Dies können sein:

- X-Koordinate
- Y-Koordinate
- Linie
- Kreis

Eine Funktion `computexy` berechnet aus den bisherigen Koordinaten und den geforderten Ortskurven die neuen Koordinaten, falls dies möglich ist. Die Berechnung kann z.B. nicht möglich sein, wenn z.B. ein Kreis keinen Schnittpunkt mit der der anderen Ortskurve hat. Gibt es für einen Punkt nur eine Ortskurve, wird das Lot von der bisherigen Position auf diesen Kreis bzw. diese Linie gebildet.

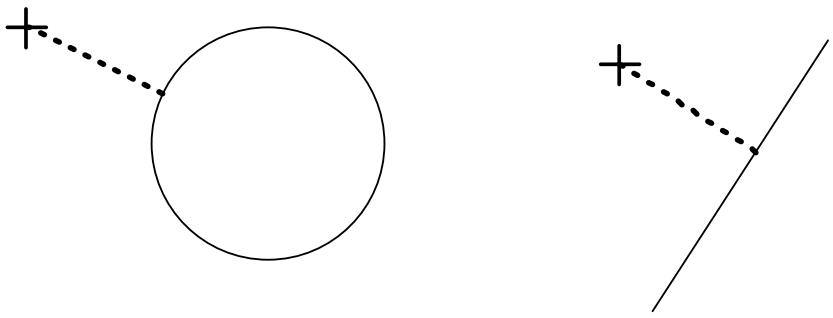


Abbildung 12

Bei den Constraint- Knoten existiert für jede Klasse eine spezielle Methode. Diese Methoden sollen für einige Klasen beschrieben werden.

Twaagrecht: Der Knoten hat eine einlaufende Kante. Die Y- Koordinate dieses Punktes bestimmt die Y- Koordinate aller Knoten, der auslaugenden Kanten.

Tsenkrecht: Analog.

Tlinie: Die 2 Knoten der einlaufenden Kanten, bestimmen die Hesse-Normalform (a, b, c) der definierten Linie. Diese Linie wird für die Knoten der auslaufenden Kanten eine Ortskurve.

Tdistanz: Dieser Knoten hat 3 adjazente Variablen- Knoten. Der Wertnoten enthält den Abstand, die beiden anderen die Punkte. Ist der Wertknoten auslaufend, so wird aus den beiden Punkten der Abstand berechnet. Man hat dann ein gesteuertes Maß. Der Wert des Maßes wird bei einer Änderung jeweils nachgeführt. Ist er einlaufend, bestimmt er und ein Punkt einen Kreis, auf dem der andere Punkt liegen muss. Dann hat man ein steuerndes Maß.

Tdistanzx, Tdistanzy: Analog.

Twinkel: Dieser Knoten hat 3 adjazente Variablen- Knoten. Der Werknoten enthält den Winkel, die beiden anderen die Punkte. Ist der Wertknoten auslaufend, so wird aus den beiden Punkten der Winkel berechnet. Ist er einlaufend, bestimmt er und ein Punkt einen Linie, auf der der andere Punkt liegen muss.

Twinkel3P: Dieser Knoten hat 4 adjazente Variablen- Knoten. Der Wertnoten enthält den Abstand, die 3 anderen die Punkte. Ist der Wertknoten auslaufend, do wird aus den 3 Punkten der Winkel berechnet. Ist er einlaufend, bestimmt er und zwei Punkte eine Ortskurve, auf der der andere Punkt liegen muss.

Tdistanzlp: Dieser Knoten hat 4 adjazente Variablen- Knoten. Der Wertnoten enthält den Abstand, die 3 anderen die Punkte. Ist der Wertknoten auslaufend, do wird aus den 3 Punkten der Abstand berechnet. Ist er einlaufend, bestimmt er und zwei Punkte eine Ortskurve, auf der der andere Punkt liegen muss.

Trechtwinklig: Analog wie Twinkel3P, jedoch ist der Winkel auf 90° festgelegt.

Tkreis: Sind die Punkte, die aus dem Kreis liegen müssen, alle durch auslaufende Kanten verbunden, wird für sie aus Mittelpunkt und Radius ein Kreis als Ortskurve bestimmt. Sind 3 Punkte mit einlaufender Kante gegeben, müssen Mittelpunkt (2x) und Radius auslaufend orientiert sein. Der Mittelpunkt und der Radius werden aus den 3 Punkten berechnet. Sind ein oder 2 Punkte auf dem Kreis einlaufend orientiert, so wird für Radius und/oder Mittelpunkt Wert bzw. Ortskurve gegeben.

Tgleichmx: 2 Punkte, die durch einlaufende Kanten verbunden sind, bestimmen die X-Koordinaten der übrigen Punkte. Bei nur einem Punkt mit einlaufender Kante, wird der erste Punkt (ganz links) zusätzlich die X- Koordinaten.

Tgleichmy: Analog.

Toperator: Aus 2 einlaufenden Wertknoten wird der Wert des anderen Knotens berechnet. Entweder aus 2 Operanden das Ergebnis oder aus dem Ergebnis und einem Operanden den anderen Operanden.

Liegt ein Zyklus vor, konvergiert die Iteration jedoch häufig nicht. Daher wird versucht für einige häufig vorkommende Fälle eine direkte Lösung zu finden.

(1) Kreis und sein Mittelpunkt

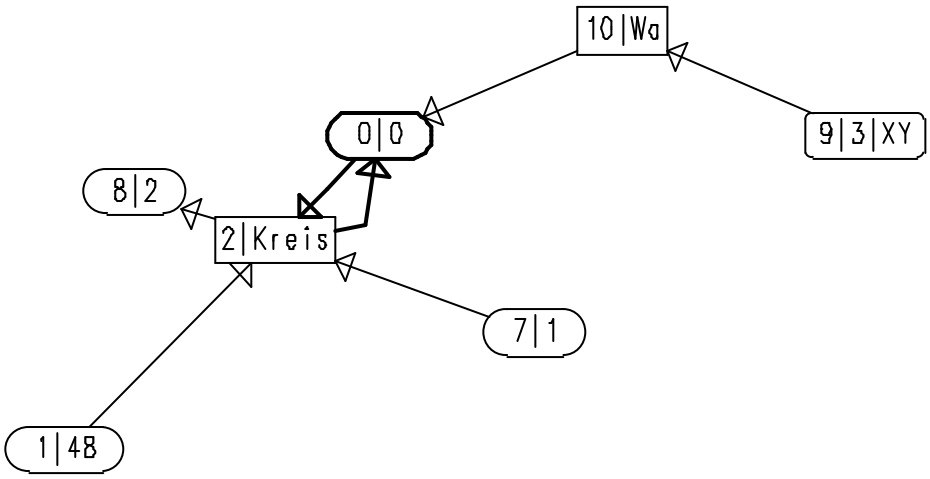


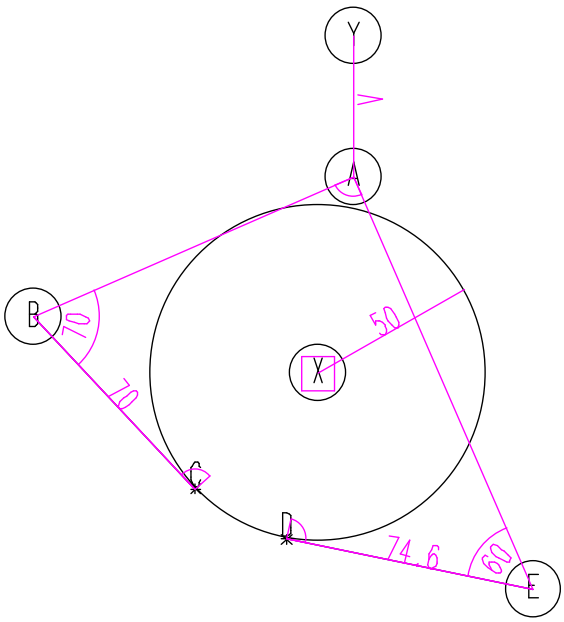
Abbildung 13

Der Kreis (3 Freiheitsgrade)ist gegeben durch:

- eine Ortlinie für den Mittelpunkt
- einen Punkt auf dem Kreis
- den Radius.

Die zweite Ortkurve für den Mittelpunkt ist durch den Kreis gegeben. Die Lage des Mittelpunktes kann berechnet werden.

(2) Winkelkette



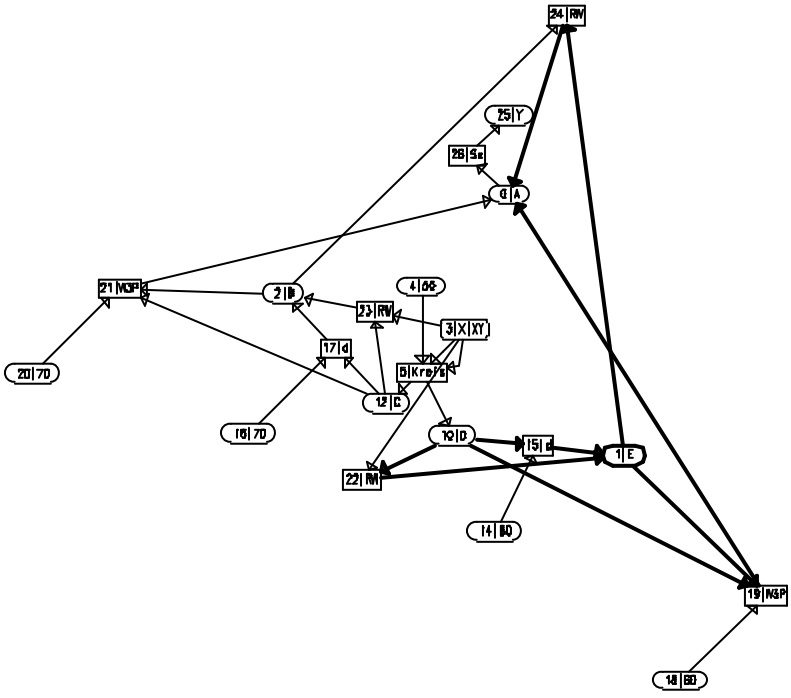


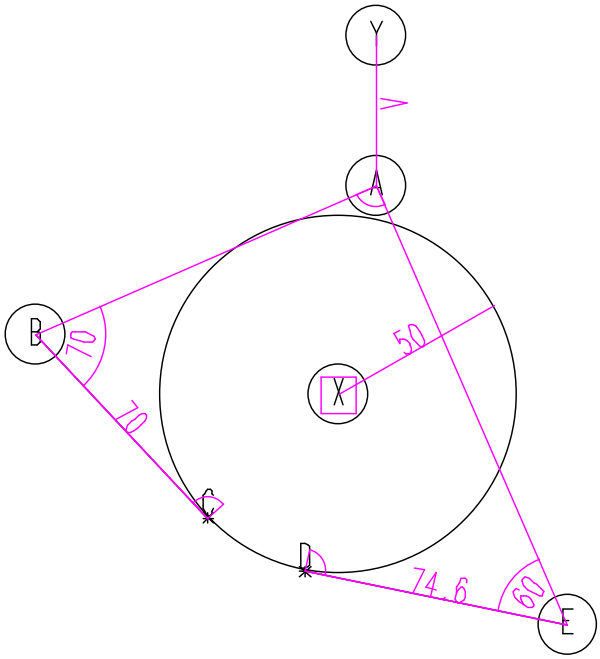
Abbildung 14

Wird z.B. der Abstand D-E geändert:

die Winkelbeziehungen zwischen D-E-A ändern sich nicht.
Damit ist die Ortslinie für A von E aus gegeben.

(3) Rigid

Als Rigid bezeichnet man eine Figur, deren Form festliegt. Bilden eine Menge von Knoten einen Rigid, so können sie nur als ganzes verschoben oder gedreht werden.
Beispiel: Verschieben von E



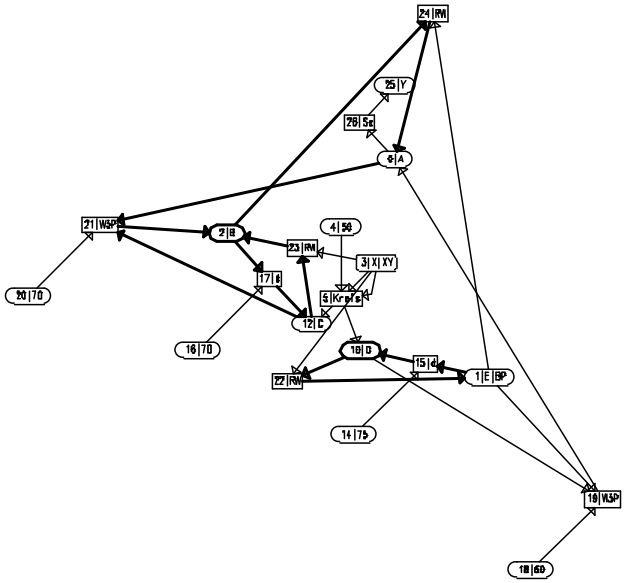
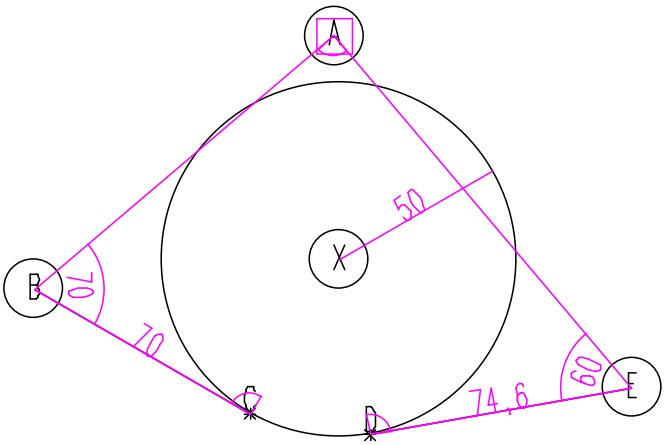


Abbildung 15

Zyklen: E, D bilden ein Rigid C, B, A müssen ihre Form nicht ändern
=> A, B, C, D, E können zusammen um X gedreht werden.

(4) Fixieren der Punkte zeitweilig aufheben



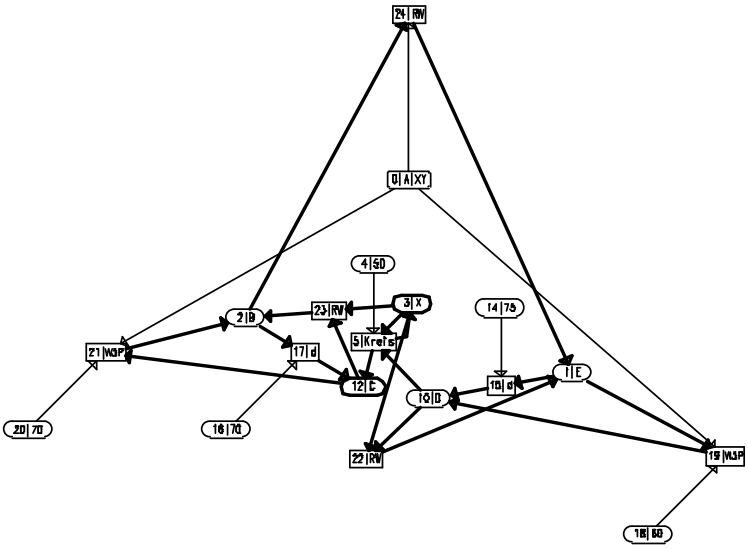


Abbildung 16

Der Abstand DE soll geändert werden.

Entfernt man die Fixierung von A:

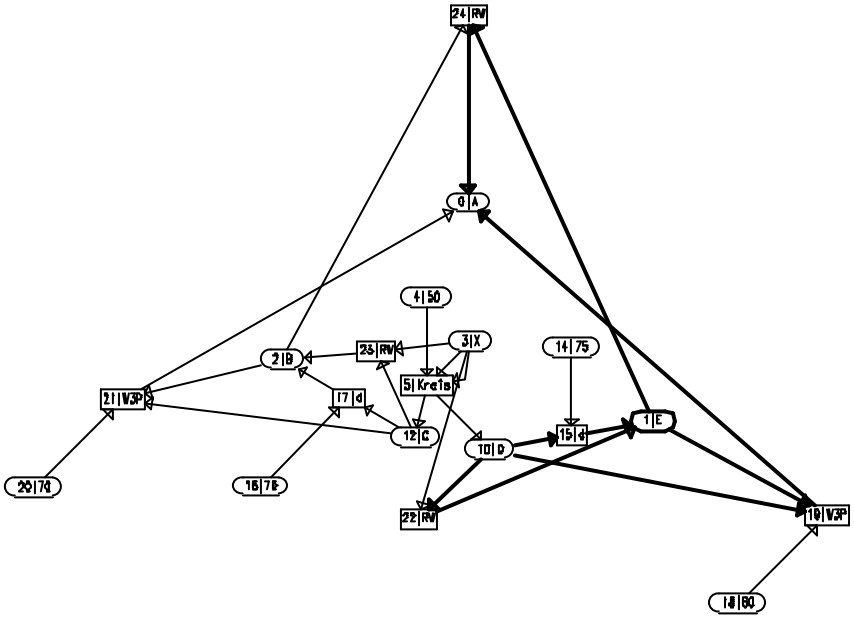


Abbildung 17

Nach der Berechnung werden alle Punkte so verschoben, dass A seine alte Position bekommt
Sind zwei Punkte fixiert, wird zwischen Ihnen ein Abstand- Constraint eingefügt.
Nach der Berechnung muss die Figur so gedreht und verschoben werden, dass beide Punkte die alte Position bekommen. Es darf dann jedoch kein Constraint existieren, das eine Richtung vorgibt.

Literatur

[Berling96] Berling R. 'Eine Constraint-basierte Modellierung für Geometrische Objekte', Dissertation.

[Du, Rosendahl, Berling 1993] Du C, Rosendahl M ,Berling R, 'Variation of Geometry and Parametric Design', Proc. 3rd. international conference on CAD and computer graphics, Beijing, Aug. 23-26, 1993, pp 400-405,international academic publishers, 1993 Du,Rosendahl,Berling1993

[Fudos, Hoffman, 1993] I. Fudos and C. M. Hoffmann. Correctness proof of a geometric constraint solver. Technical Report 93-076, Purdue University, Computer Science, 1993. Correctness Proof of a Geometric Constraint Solver

[Hoffmann89] C.M. Hoffmann. Solid and Geometric Modeling. Morgan Kaufmann, 1989.

[Hsu, Brüderlin 1997] C. Hsu, B. Brüderlin. "A Hybrid Constraint Solver Using Exact and Iterative Geometric Constructions," in CAD Systems Development -- Tools and Methods, D. Roller, P. Brunet, eds., Springer Verlag, 1997.

[Kondo, 1992] K. Kondo. Algebraic method for manipulation of dimensional relationships in geometric models. Computer Aided Design, 24(3):141--147, March 1992.

[Light, Gossard 1982] R. Light and D. Gossard. Modification of geometric models through variational geometry. Computer Aided Design, 14:209--214, July 1982

[Verroust, Schonek, Roller, 1992] A. Verroust, F. Schonek, and D. Roller. Rule-oriented method for parameterized Computer-aided design. Computer Aided Design, 24 (10):531-540, October 1992.

Available Research Reports (since 1999):

2004

12/2004 *Manfred Rosendahl.* Objektorientierte Implementierung einer Constraint basierten geometrischen Modellierung.

11/2004 *Urs Kuhlmann, Harry Sneed, Andreas Winter.* Workshop Reengineering Prozesse (RePro 2004) — Fallstudien, Methoden, Vorgehen, Werkzeuge.

10/2004 *Bernhard Beckert, Gerd Beuster.* Formal Specification of Security-relevant Properties of User-Interfaces.

9/2004 *Bernhard Beckert, Martin Giese, Elmar Habermalz, Reiner Hähnle, Andreas Roth, Philipp Rümmer, Steffen Schlager.* Taclets: A New Paradigm for Constructing Interactive Theorem Provers.

8/2004 *Achim Rettinger.* Learning from Recorded Games: A Scoring Policy for Simulated Soccer Agents.

7/2004 *Oliver Obst, Markus Rollmann.* Spark — A Generic Simulator for Physical Multi-agent Simulations.

6/2004 *Frank Dylla, Alexander Ferrein, Gerhard Lakemeyer, Jan Murray, Oliver Obst, Thomas Röfer, Frieder Stolzenburg, Ubbo Visser, Thomas Wagner.* Towards a League-Independent Qualitative Soccer Theory for RoboCup.

5/2004 *Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt, Thomas Kleemann.* Model Based Deduction for Database Schema Reasoning.

4/2004 *Lutz Priese.* A Note on Recognizable Sets of Unranked and Unordered Trees.

3/2004 *Lutz Priese.* Petri Net DAG Languages and Regular Tree Languages with Synchronization.

2/2004 *Ulrich Furbach, Margret Groß-Hardt, Bernd Thomas, Tobias Weller, Alexander Wolf.* Issues Management: Erkennen und Beherrschen von kommunikativen Risiken und Chancen.

1/2004 *Andreas Winter, Carlo Simon.* Exchanging Business Process Models with GXL.

2003

18/2003 *Kurt Lautenbach.* Duality of Marked Place/Transition Nets.

17/2003 *Frieder Stolzenburg, Jan Murray, Karsten Sturm.* Multiagent Matching Algorithms With and Without Coach.

16/2003 *Peter Baumgartner, Paul A. Cairns, Michael Kohlhase, Erica Melis (Eds.).* Knowledge Representation and Automated Reasoning for E-Learning Systems.

15/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, Thomas Kleemann, Christoph Wernhard.* KRHyper Inside — Model Based Deduction in Applications.

14/2003 *Christoph Wernhard.* System Description: KRHyper.

13/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, Alex Sinner.* 'Living Book' :- 'Deduction', 'Slicing', 'Interaction'..

12/2003 *Heni Ben Amor, Oliver Obst, Jan Murray.* Fast, Neat and Under Control: Inverse Steering Behaviors for Physical Autonomous Agents.

11/2003 *Gerd Beuster, Thomas Kleemann, Bernd Thomas.* MIA - A Multi-Agent Location Based Information Systems for Mobile Users in 3G Networks.

10/2003 *Gerd Beuster, Ulrich Furbach, Margret Groß-Hardt, Bernd Thomas.* Automatic Classification for the Identification of Relationships in a Metadata Repository.

9/2003 *Nicholas Kushmerick, Bernd Thomas.* Adaptive information extraction: Core technologies for information agents.

8/2003 *Bernd Thomas.* Bottom-Up Learning of Logic Programs for Information Extraction from Hypertext Documents.

7/2003 *Ulrich Furbach.* AI - A Multiple Book Review.

6/2003 *Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt.* Living Books.

5/2003 *Oliver Obst.* Using Model-Based Diagnosis to Build Hypotheses about Spatial Environments.

4/2003 *Daniel Lohmann, Jürgen Ebert.* A Generalization of the Hyperspace Approach Using Meta-Models.

3/2003 *Marco Kögler, Oliver Obst.* Simulation League: The Next Generation.

2/2003 *Peter Baumgartner, Margret Groß-Hardt, Alex Sinner.* Living Book – Deduction, Slicing and Interaction.

1/2003 *Peter Baumgartner, Cesare Tinelli.* The Model Evolution Calculus.

2002

12/2002 *Kurt Lautenbach.* Logical Reasoning and Petri Nets.

11/2002 *Margret Groß-Hardt.* Processing of Concept Based Queries for XML Data.

10/2002 *Hanno Binder, Jérôme Diebold, Tobias Feldmann, Andreas Kern, David Polock, Dennis Reif, Stephan Schmidt, Frank Schmitt, Dieter Zöbel.* Fahrassistenzsystem zur Unterstützung beim Rückwärtsfahren mit einachsigen Gespannen.

9/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 4. Workshop Software Reengineering (Bad Honnef, 29./30. April 2002).

8/2002 *Richard C. Holt, Andreas Winter, Jingwei Wu.* Towards a Common Query Language for Reverse Engineering.

7/2002 *Jürgen Ebert, Bernt Kullbach, Volker Riediger, Andreas Winter.* GUPRO – Generic Understanding of Programs, An Overview.

6/2002 *Margret Groß-Hardt.* Concept based querying of semistructured data.

5/2002 *Anna Simon, Marianne Valerius.* User Requirements – Lessons Learned from a Computer Science Course.

4/2002 *Frieder Stolzenburg, Oliver Obst, Jan Murray.* Qualitative Velocity and Ball Interception.

3/2002 *Peter Baumgartner.* A First-Order Logic Davis-Putnam-Logemann-Loveland Procedure.

2/2002 *Peter Baumgartner, Ulrich Furbach.* Automated Deduction Techniques for the Management of Personalized Documents.

1/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 3. Workshop Software Reengineering (Bad Honnef, 10./11. Mai 2001).

2001

13/2001 *Annette Pook.* Schlussbericht “FUN - Funkunterrichtsnetzwerk”.

12/2001 *Toshiaki Arai, Frieder Stolzenburg.* Multiagent Systems Specification by UML Statecharts Aiming at Intelligent Manufacturing.

11/2001 *Kurt Lautenbach.* Reproducibility of the Empty Marking.

10/2001 *Jan Murray.* Specifying Agents with UML in Robotic Soccer.

9/2001 *Andreas Winter.* Exchanging Graphs with GXL.

8/2001 *Marianne Valerius, Anna Simon.* Slicing Book Technology — eine neue Technik für eine neue Lehre?.

7/2001 *Bernt Kullbach, Volker Riediger.* Folding: An Approach to Enable Program Understanding of Preprocessed Languages.

6/2001 *Frieder Stolzenburg.* From the Specification of Multiagent Systems by Statecharts to their Formal Analysis by Model Checking.

5/2001 *Oliver Obst.* Specifying Rational Agents with Statecharts and Utility Functions.

4/2001 *Torsten Gipp, Jürgen Ebert.* Conceptual Modelling and Web Site Generation using Graph Technology.

3/2001 *Carlos I. Chesñevar, Jürgen Dix, Frieder Stolzenburg, Guillermo R. Simari.* Relating Defeasible and Normal Logic Programming through Transformation Properties.

2/2001 *Carola Lange, Harry M. Sneed, Andreas Winter.* Applying GUPRO to GEOS – A Case Study.

1/2001 *Pascal von Hutten, Stephan Philippi.* Modelling a concurrent ray-tracing algorithm using object-oriented Petri-Nets.

2000

8/2000 *Jürgen Ebert, Bernt Kullbach, Franz Lehner (Hrsg.).* 2. Workshop Software Reengineering (Bad Honnef, 11./12. Mai 2000).

7/2000 *Stephan Philippi.* AWPN 2000 - 7. Workshop Algorithmen und Werkzeuge für Petrinetze, Koblenz, 02.-03. Oktober 2000 .

6/2000 *Jan Murray, Oliver Obst, Frieder Stolzenburg.* Towards a Logical Approach for Soccer Agents Engineering.

5/2000 *Peter Baumgartner, Hantao Zhang (Eds.).* FTP 2000 – Third International Workshop on First-Order Theorem Proving, St Andrews, Scotland, July 2000.

4/2000 *Frieder Stolzenburg, Alejandro J. García, Carlos I. Chesñevar, Guillermo R. Simari.* Introducing Generalized Specificity in Logic Programming.

3/2000 *Ingar Uhe, Manfred Rosendahl.* Specification of Symbols and Implementation of Their Constraints in JKogge.

2/2000 *Peter Baumgartner, Fabio Massacci.* The Taming of the (X)OR.

1/2000 *Richard C. Holt, Andreas Winter, Andy Schürr.* GXL: Towards a Standard Exchange Format.

1999

10/99 *Jürgen Ebert, Luuk Groenewegen, Roger Süttenbach.* A Formalization of SOCCA.

9/99 *Hassan Diab, Ulrich Furbach, Hassan Tabbara.* On the Use of Fuzzy Techniques in Cache Memory Managament.

8/99 *Jens Woch, Friedbert Widmann.* Implementation of a Schema-TAG-Parser.

7/99 *Jürgen Ebert, and Bernt Kullbach, Franz Lehner (Hrsg.).* Workshop Software-Reengineering (Bad Honnef, 27./28. Mai 1999).

6/99 *Peter Baumgartner, Michael Kühn.* Abductive Coreference by Model Construction.

5/99 *Jürgen Ebert, Bernt Kullbach, Andreas Winter.* GraX – An Interchange Format for Reengineering Tools.

4/99 *Frieder Stolzenburg, Oliver Obst, Jan Murray, Björn Bremer.* Spatial Agents Implemented in a Logical Expressible Language.

3/99 *Kurt Lautenbach, Carlo Simon.* Erweiterte Zeitstempelnetze zur Modellierung hybrider Systeme.

2/99 *Frieder Stolzenburg.* Loop-Detection in Hyper-Tableaux by Powerful Model Generation.

1/99 *Peter Baumgartner, J.D. Horton, Bruce Spencer.* Merge Path Improvements for Minimal Model Hyper Tableaux.