Parametrische Modellierung

In CAD Systemen möchte man nicht nur die geometrischen Objekte definieren sondern auch funktionale Abhängigkeiten zwischen den geometrischen Objekten und auch sonstigen Variablen, z.B. Materialkonstanten oder Belastungen definieren.

Dies geschieht durch **Constraints**. Dies kann sein:

- Abstand von 2 Punkten
- 2 Linien gleichlang oder parallel.
- Winkel zwischen 3 Punkten
- Punkte horizontal oder vertikal ausgerichtet
- Punkte auf einer Linie
- X und/oder Y Koordinate von Punkten

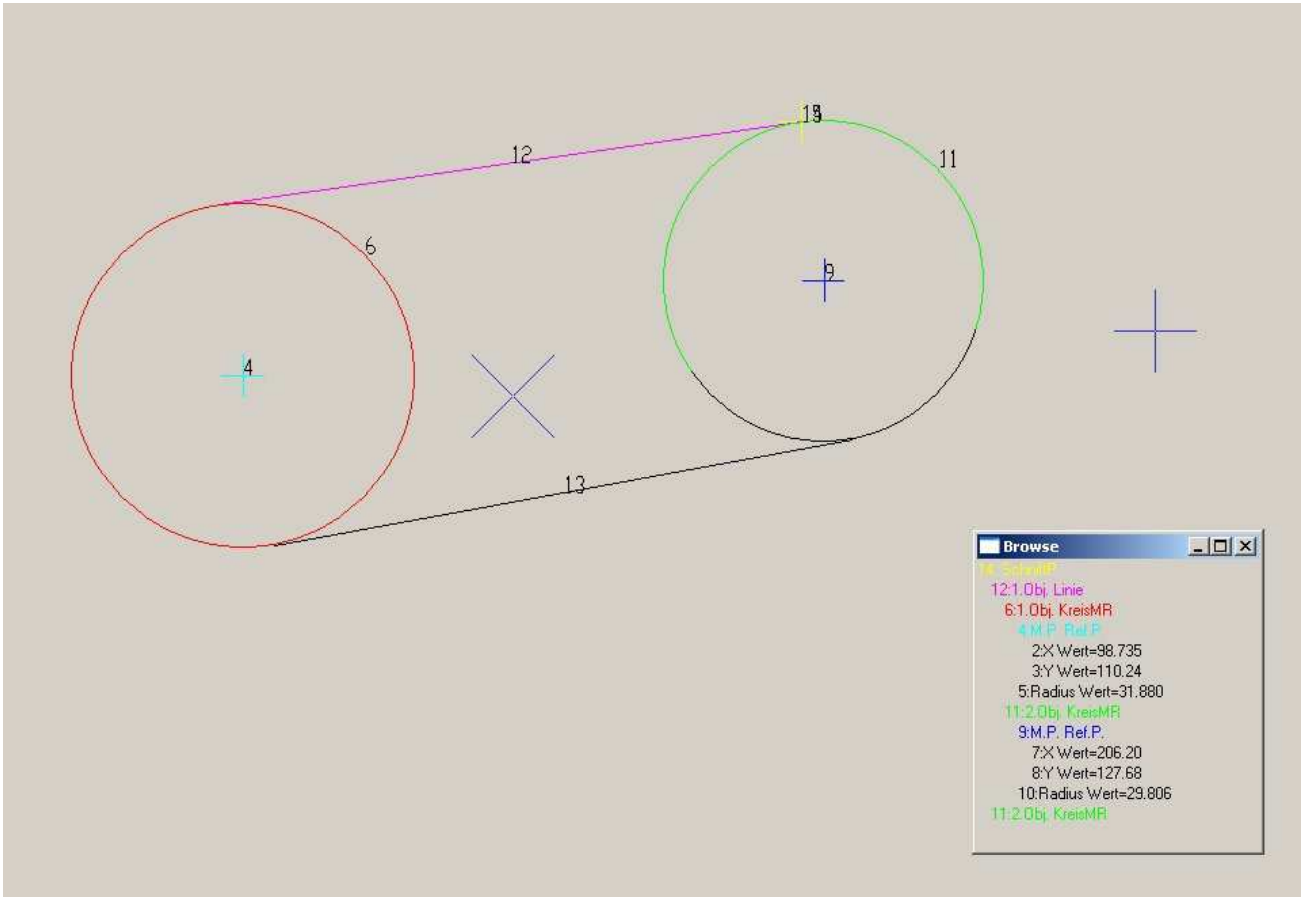
Die Techniken zur Lösung dieser Geometrischen Constraint Systeme können in 3 Klassen eingeteilt werden:

- 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.
  Regelbasiert [Verroust, Schonek, Roller, 1992], [Fudos, Hoffman, 1993], [Fudos, Hoffmann 1997] , [Fudos 2000]
  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. Ein anderer Regelbasierter Ansatz beruht auf logischen Regeln [Aldefeld 88] [Brüderlin93]

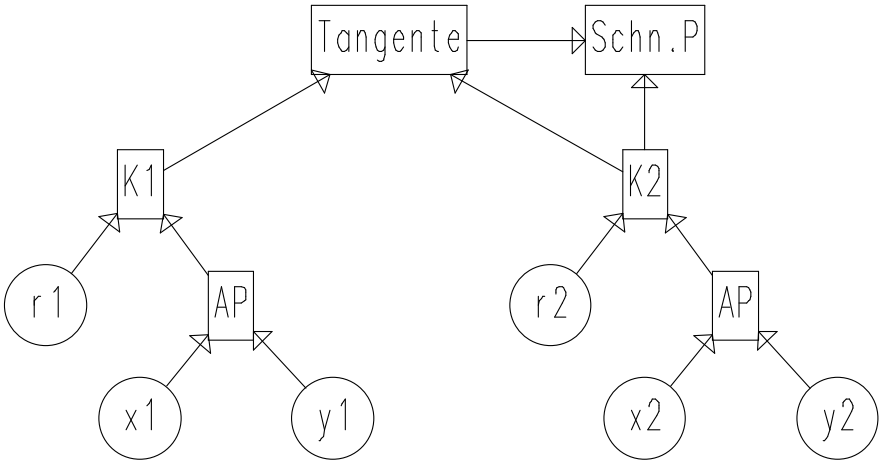
- **Graphbasiert**
  Graphbasierte Techniken modellieren die Constraints als Hyperkanten zwischen der geometrischen oder sonstigen Variablen. Statt mit einem Hypergraphen, kann man auch mit einem bipartiten Graphen arbeiten, bei dem jeweils Constraints und Variablen durch zunächst ungerichtete Kanten verbunden sind
  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.
  [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.

In **Parametrischen** Systemen werden die Abhängigkeiten durch einen **DAG** dargestellt. Damit können dann allerdings keine Symmetrischen Beziehungen, wie etwa ‚2 Linien sind gleich lang, definiert werden. Es kann nur gefordert werden, dass

die 2. Linie ihre Länge von der ersten übernimmt. Die Blätter dieses DAG sind Skalare Werte.

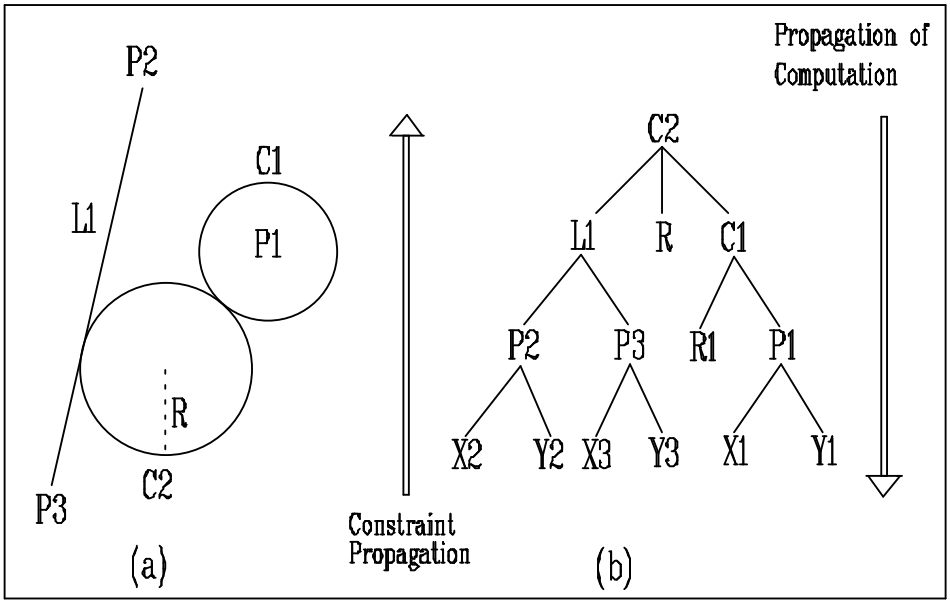


Der Graph dazu:



Bei einer Änderung des Wertes in einem Blatt kann mit Hilfe der topologischen Sortierung, der DAG neu berechnet werden. Diese toplogische Sortierung kann z.B.

dadurch erfolgen, dass für jede Klasse eine Methode `compute` definiert ist, die zunächst `compute` für die referierten Elemente aufruft und dann neu berechnet. Ferner wird noch festgehalten, ob ein Knoten schon neu berechnet worden ist. Für die Blätter ist `compute` leer.



Der Kreis C2 hat den Radius R und liegt tangential an der Linie L1 und an dem Kreis C1. Bei der Berechnung von C2 wird zunächst L1 und C1 aus den Punkten P1, P2 und P3 berechnet.

Will man z.B. einen Punkt als Schnittpunkt von 2 Linien definieren, ist es unerheblich, wie die beiden Linien entstanden sind, maßgeblich sind nur die Koordinaten der Linien. Daher werden zunächst 5 so genannte abstrakte Klassen definiert:

```
line = object(item)
  x1,y1,x2,y2:real;
  constructor init(xx1,yy1,xx2,yy2:real);
  procedure draw;virtual;
end;

circle = object(item)
  x,y,r:real;
  constructor init(xx,yy,rr:real);
  procedure draw;virtual;
end;

arc = object(circle)
  phi1,phi2:real;
  constructor init(xx,yy,rr,iphi1,iphi2:real);
  procedure draw;virtual;
end;
```

```
value=object(any)
 x:real;
 constructor init(xx:real);
end;

point=object(any)
 x,y:real;
 constructor init(xx,yy:real);
 procedure draw;virtual;
end;
```

Alle in einem Modell vorkommenden Elemente sind Instanzen von Klassen, die aus diesen absoluten Klassen abgeleitet werden.
Punkte können z.B. wie folgt definiert werden:

- **Punkt, der sich auf 2 Skalare Werte bezieht**

```
pointref=object(point)
 irx,iry:integer;
 function rx:valueptr;
 function ry:valueptr;
 procedure compute(kind:tvalkind);virtual;
 constructor init(rrx,rry:valueptr);
end;
```

Bei dieser Implementation sind die alle Elemente in eine Liste eingetragen. Der Bezug ist jeweils der Index in der Liste.

- **Punkt, der relativ zu einem anderen Punkt mit einem Abstand, der durch 2 Skalare gegeben ist.**

```
pointrel=object(point)
 irdx,irdy,ip1:integer;
 function rdx:valueptr;
 function rdy:valueptr;
 function p1:pointptr;
 procedure compute(kind:tvalkind);virtual;
 constructor init(pp:pointptr;rrdx,rrdy:valueptr);
end;
```

- **Punkt definiert als Schnittpunkt von 2 Elementen. Da es evtl. mehrere Schnittpunkte gibt ist der Index des Schnittpunktes angegeben.**

```
pointinters=object(point)
{ alter: 0,1 on circle}
 ii1,ii2:integer;
 function i1:itemptr;
 function i2:itemptr;
 procedure compute(kind:tvalkind);virtual;
 constructor init(i11,i22:itemptr;palter:integer);
 procedure load(var id:stream);virtual;
 procedure store(var id:stream);virtual;
 function number_of_relation:integer;virtual;
```

```
 function get_relation(n:integer):integer;virtual;
 procedure new_relation(n:integer;r:integer);virtual;
end;
```

- **Punkt definiert als Endpunkt einer Tangente von einem Punkt an einen Kreis**

```
pointtangente=object(point)
{ alter: 0,1 on circle}
 ip1,ic1:integer;
 function p1:pointptr;
 function c1:circleptr;
 procedure compute(kind:tvalkind);virtual;
 constructor init(pp1:pointptr;cc1:circleptr;palter:integer);
end;
```

- **Punkt definiert als Polarpunkt mit einem Bezugspunkt**

```
pointpolar=object(point)
 ip1,ir1,iphi1:integer;
 function p1:pointptr;
 function r1:valueptr;
 function phi1:valueptr;
 procedure compute(kind:tvalkind);virtual;
 constructor init(pp1:pointptr;pphi1,rr1:valueptr);
end;
```

- **Punkt definiert als Loptpunkt von einem Punkt an eine Linie**

```
pointlot=object(point)
 ip1,ii1:integer;
 function p1:pointptr;
 function i1:itemptr;
 procedure compute(kind:tvalkind);virtual;
 constructor init(pp1:pointptr;ii:itemptr);
end;
```

- **Punkt definiert als Punkt einer Tangente zwischen 2 Elementen**

```
tangente2e=object(point)
{ alter
  0,1: point is on circle, the other item is line;
  2,3: point is on line, the other item is circle;
  0-3: point is on the first circle, the other item is circle;
  4-7: point is on the second circle,the other item is circle;
}
 ii1,ii2:integer;
 function i1:itemptr;
 function i2:itemptr;
 procedure compute(kind:tvalkind);virtual;
 constructor init(i11,i22:itemptr;palter:integer);
 procedure load(var id:stream);virtual;
```

```
 procedure store(var id:stream);virtual;
 function number_of_relation:integer;virtual;
 function get_relation(n:integer):integer;virtual;
 procedure new_relation(n:integer;r:integer);virtual;
end;
```

- **Punkt definiert als Tangentialpunkt eines Kreises tangential zu 2 Elementen**

```
circle2er=object(point)
{ alter: 0-7}
 ii1,ii2,ir:integer;
 function i1:itemptr;
 function i2:itemptr;
 function r:valueptr;
 procedure compute(kind:tvalkind);virtual;
 constructor init(i11,i22:itemptr;rr:valueptr;palter:integer);
 procedure load(var id:stream);virtual;
 procedure store(var id:stream);virtual;
 function number_of_relation:integer;virtual;
 function get_relation(n:integer):integer;virtual;
 procedure new_relation(n:integer;nr:integer);virtual;
end;
```

- **Punkt, der ausgezeichneter Punkt eines Elements ist (Endpunkt, Mittelpunkt)**

```
pointitem=object(point)
{ alter:
  0: centre point;
  1: the first point;
  2: the second point;
}
 ii1:integer;
 function i1:itemptr;
 procedure compute(kind:tvalkind);virtual;
 constructor init(i11:itemptr;palter:integer);
end;
```

Ebenso können auch die anderen abstrakten Elemente, wie Linie, Kreis von anderen Elementen abhängig sein.

- **Linie, die von 2 Elementen abhängig ist**

```
line_2o = object(line)
{ alter
  0  : pp;
  0-1: dl,pl;
  0-2: pc;
  0-4: lc;
  0-4: cc;
  0-8: ll;
```

```
}
  ia1,ia2:integer;
  constructor init(aa1,aa2:anyptr;lalter:integer);
  function i1 :anyptr;
  function i2 :anyptr;
  procedure compute(kind:tvalkind);virtual;
end;
```

Die Linie kann sich beziehen auf:

- 2 Punkte (Verbindungslinie)
- Punkt und Linie (Lot von Punkt an Linie, Parallele durch Punkt zu Linie)
- Linie und Wert (Parallele mit Abstand Wert)
- Punkt und Kreis (Tangenten und Verbindung zum Mittelpunkt)
- 2 Kreise (Tangenten und Verbindung der Mittelpunkte)
- 2 Linien (Winkelhalbierende, Verbindung der Endpunkte)

Ein Kreis wird bestimmt durch 3 Elemente. Diese können sein:
X-Wert Mittelpunkt
Y-Wert Mittelpunkt
Radius
Tangentiales Element (Punkt, Linie, Kreis)

- **Kreis abhängig von 3 Elementen**

```
circle_3o = object(circle)
{ alter: 0-15}
  ii1,ii2,ii3 : integer;
  constructor init(i11,i22,i33:anyptr;ialter:integer);
  function i1 : anyptr;
  function i2 : anyptr;
  function i3 : anyptr;
  procedure compute(kind:tvalkind);virtual;
end;
```

Dasselbe ist auch für Kreisbogen möglich:

- **Bogen abhängig von 3 Elementen**

```
arc_3o = object(arc)
  ii1,ii2,ii3 : integer;
{  alter:integer;}
  constructor init(i11,i22,i33:anyptr;ialter:integer);
  function i1 : anyptr;
  function i2 : anyptr;
  function i3 : anyptr;
  procedure store(var id:stream);virtual;
  procedure load(var id:stream);virtual;
  procedure compute(kind:tvalkind);virtual;
  function number_of_relation:integer;virtual;
  function get_relation(n:integer):integer;virtual;
  procedure new_relation(n:integer;nr:integer);virtual;
```

```
end;
```

Es können nicht nur die geometrischen Elemente von den Werten oder Punkten abhängig sein, sondern auch die Werte können von den geometrischen Elementen abhängig sein.

- **Wert abhängig von einem Punkt**

```
value1p=object(value)
 ip1:integer;
 kind:tvalkind;
 function p1:pointptr;
 procedure compute(tkind:tvalkind);virtual;
 function getval:real;virtual;
 constructor init(pp1:pointptr;kkind:tvalkind);
end;
```

Der Wert ist entweder die X oder die Y Koordinate eines Punktes. Oder der Abstand vom Nullpunkt bzw. der Winkel zum Nullpunkt.

- **Wert abhängig von 2 Punkten**

```
value2p=object(value)
 ip1,ip2:integer;
 kind:tvalkind;
 function p1:pointptr;
 function p2:pointptr;
 procedure compute(tkind:tvalkind);virtual;
 function getval:real;virtual;
 constructor init(pp1,pp2:pointptr;kkind:tvalkind);
end;
```

Der Wert die die Differenz der X bzw. Y Koordinaten oder der Abstand der beiden Punkte bzw. der Winkel von 1. zum 2. Punkt.

- **Wert abhängig von einer Linie**

```
value_line = object(value)
  il1 : integer;
  kind: tvalkind;
  constructor init(ii1:lineptr;ikind:tvalkind);
  function i1:lineptr;
  function getval:real;virtual;
  procedure compute(tkind:tvalkind);virtual;
end;
```

Hier ist der Wert ähnlich wie bei 2 Punkten, es werden der Anfangs- und der Endpunkt der Linie genommen.

- **Wert abhängig von einem Kreis**

```
value_circle = object(value)
  ic1 : integer;
  kind: tvalkind;
  constructor init(ii1:circleptr;ikind:tvalkind);
  function i1:circleptr;
  function getval:real;virtual;
  procedure compute(tkind:tvalkind);virtual;
end;
```

Der Wert bezieht sich auf den Radius oder auf den Mittelpunkt, ähnlich wie beim Wert von einem Punkt

- **Wert abhängig von einem Bogen**

```
value_arc = object(value)
  ia1 : integer;
  kind: tvalkind;
  constructor init(ii1:arcptr;ikind:tvalkind);
  function i1:arcptr;
  function getval:real;virtual;
  procedure compute(tkind:tvalkind);virtual;
end;
```

Ähnlich wie beim Wert eines Kreises. Zusätzlich kann noch der Anfangs- und der Endwinkel ermittelt werden.

- **Wert abhängig von 2 Werten (Operator)**

```
exp=object(value)
 op:top;
 il,ir:integer;
 function l:valueptr;
 function r:valueptr;
 procedure compute(kind:tvalkind);virtual;
 function getval:real;virtual;
 constructor init(oop:top;ll,rr:valueptr);
end;
```

Hier werden die Werte l und r durch den Operator op verknüpft. Damit lassen sich arithmetische Ausdrücke aufbauen. Für Operatoren oder Funktionen mit nur einem Argument gibt es:

- **Wert abhängig von einem Wert (Funktion)**

```
func=object(value)
 op:top;
 iv:integer;
 function v:valueptr;
 procedure compute(kind:tvalkind);virtual;
 function getval:real;virtual;
 constructor init(oop:top;vv:valueptr);
end;
```

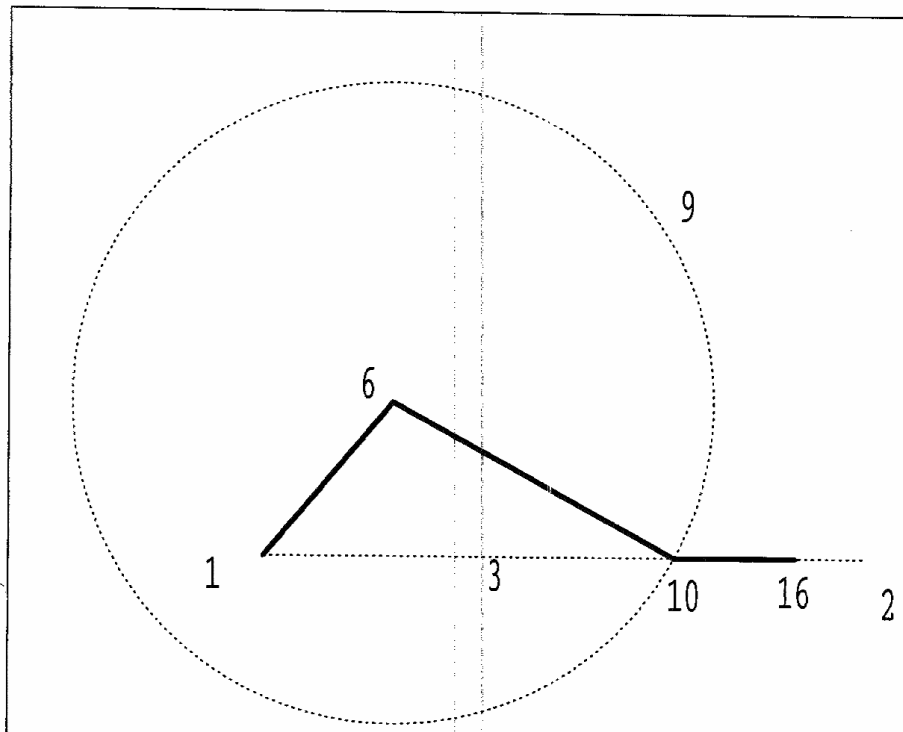
Hier wird auf den Wert v der Operator bzw. die Funktion op angewandt. Die Funktionen sind:
Negation, Log, Sin, Cos.

Klassenhierachie:

```
-Base
 +--Node
    +--any
    ¦  +--dummy
    ¦  +--instance
    ¦  ¦  +--association
    ¦  ¦  +--instanceseg
    ¦  +--item
    ¦  ¦  +--circle
    ¦  ¦  ¦  +--arc
    ¦  ¦  ¦  ¦  +--arcseg
    ¦  ¦  ¦  ¦  +--arc_3o
    ¦  ¦  ¦  ¦  +--arc_cr2o
    ¦  ¦  ¦  ¦  +--arc_r2o
    ¦  ¦  ¦  +--circleseg
    ¦  ¦  ¦  +--circle_3o
    ¦  ¦  ¦  +--circle_co
    ¦  ¦  ¦  +--circle_r2o
    ¦  ¦  ¦  +--entity
    ¦  ¦  +--line
    ¦  ¦  ¦  +--lineseg
    ¦  ¦  ¦  +--line_2o
    ¦  ¦  +--marker
    ¦  +--lateconstraint
    ¦  +--point
    ¦  ¦  +--circle2er
    ¦  ¦  +--pointinters
    ¦  ¦  +--pointitem
    ¦  ¦  +--pointlot
    ¦  ¦  +--pointpolar
    ¦  ¦  +--pointref
    ¦  ¦  +--pointrel
    ¦  ¦  +--pointrelref
    ¦  ¦  +--pointseg
    ¦  ¦  +--pointtangente
    ¦  ¦  +--tangente2e
    ¦  +--segment
    ¦  +--value
    ¦     +--exp
    ¦     +--func
    ¦     +--value1p
    ¦     +--value2p
    ¦     +--value_arc
    ¦     +--value_circle
    ¦     +--value_line
```

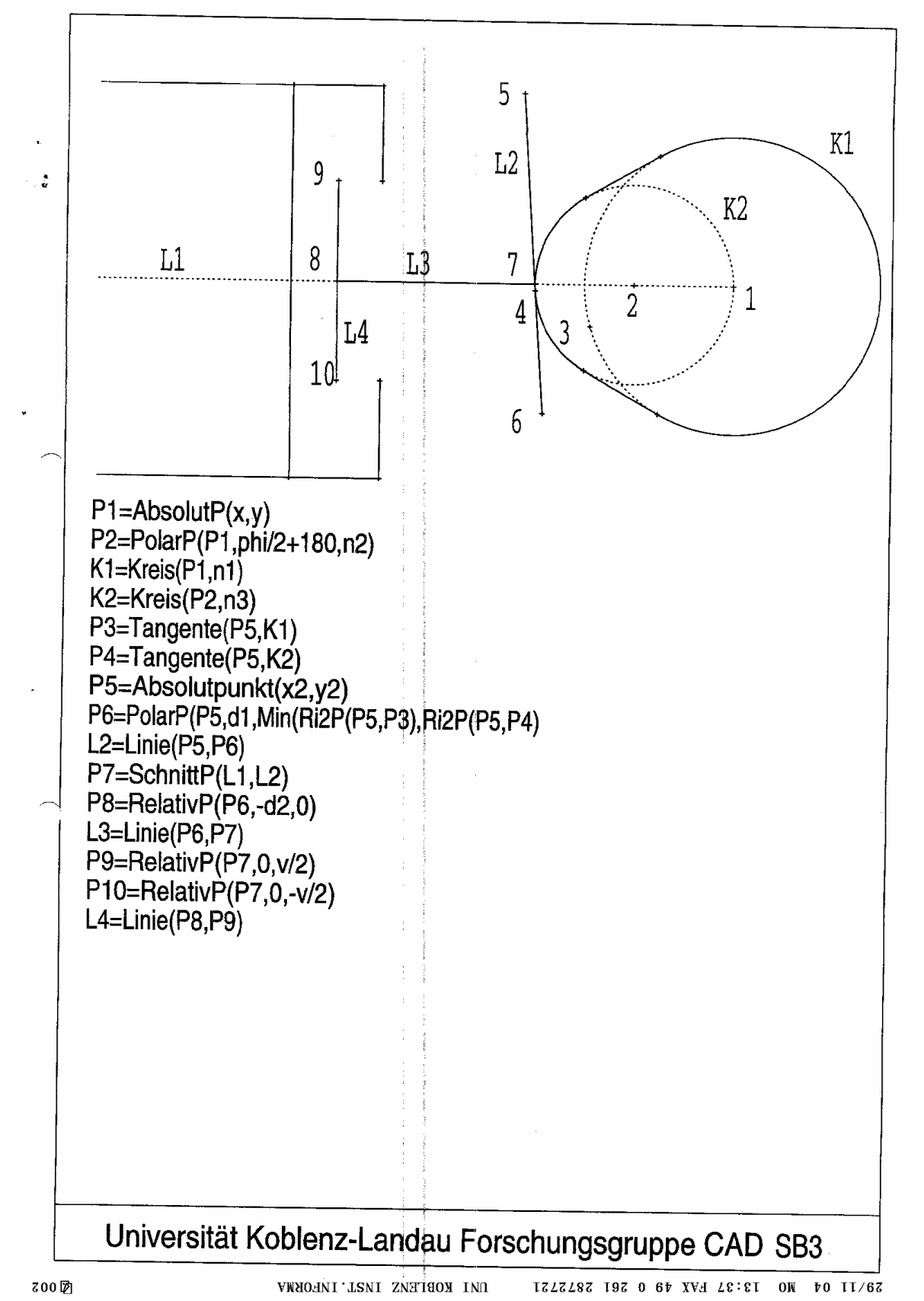
```
    +--anynode
    +--orefnode
```

Beispiel: Schema eines Motors



1:	Punkt	X:	100	Y:	100			
2:	Punkt	X:	250	Y:	100			
3:	Linie	P1:	1	P2:	2			
4:	Wert	Wert:	50					
5:	Wert	Wert:	50					
6:	PPolar	RefP:	1	Abstan:	5	Winkel:	4	
7:	Linie	P1:	1	P2:	6			
8:	Wert	Wert:	80					
9:	Kreis	MitP:	6	Radius:	8			
10:	PSchni	Altern:	1	Item1:	9	Item2:	3	
11:	Linie	P1:	6	P2:	10			
12:	Wert1P	P1:	10	Typ:	X-WERT	Wert:	202.374	
13:	Wert	Wert:	30					
14:	Ausdru	Op:	+	LOp:	12	ROp:	13	Wert: 232.374
15:	Wert1P	P1:	10	Typ:	Y-WERT	Wert:	100	
16:	PRef	RefX:	14	RefY:	15			
17:	Linie	P1:	10	P2:	16			

Universität Koblenz-Landau Forschungsgruppe CAD SB1



Vorteil der parametrischen Modellierung ist, dass man den DAG immer direkt evaluieren kann. Da die einzelnen Klassen nur jeweils so viele bestimmende

Elemente haben, wie das zu erzeugende Element Freiheitsgrade hat, sind die Konstruktionen nie unter- oder überbestimmt.
Es ist jedoch nicht einfach möglich, etwa für die Lage eines Schnittpunktes eine neue Koordinate vorzugeben und die sich schneidenden Linien entsprechend zu verändern. Ferner kann man nicht nachträglich zusätzliche Constraints (Randbedingungen) einfügen, es sei denn man hat ein Element einer abstrakten Klasse vorliegen, das durch ein Element der abgeleiteten Klasse ersetzt wird.

Im DAG sind nur die Verweise von den Elementen zu den Blättern gespeichert. Möchte man prüfen, ob ein zwischen zwei Elementen eine Abhängigkeit besteht, kann man die rekursive Prozedur zur Neuberechnung nutzen. Statt jeweils die Neuberechnung durchzuführen wird nur der erreichte Knoten markiert. Wird bei der Berechnung von Knoten A Knoten B markiert, ist A von B abhängig.

So können alle Wege im Graphen ermittelt werden.

Manche Konstruktionen sind allerdings überhaupt nicht in Form eines Graphen ohne Zyklen zu modellieren.

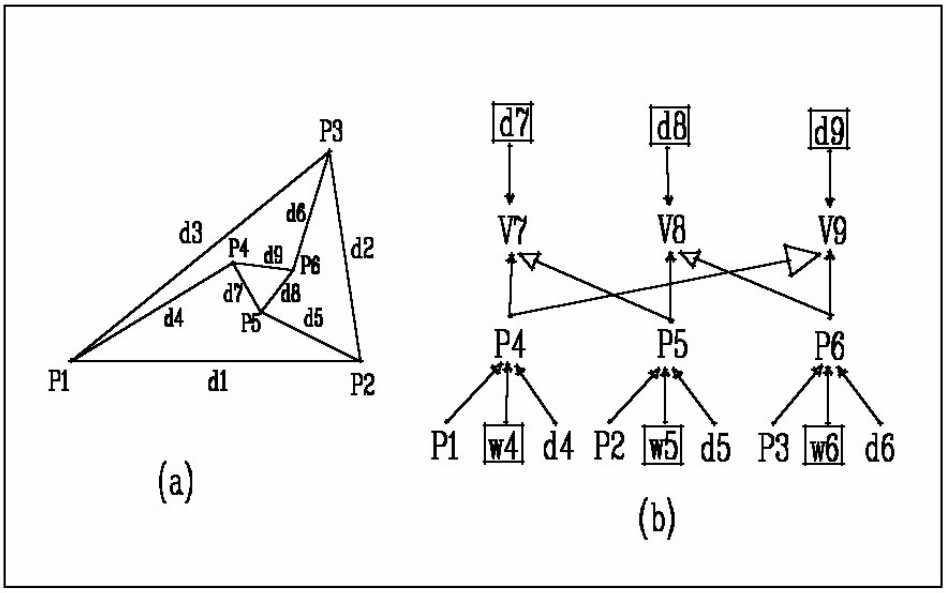


Abb. 5

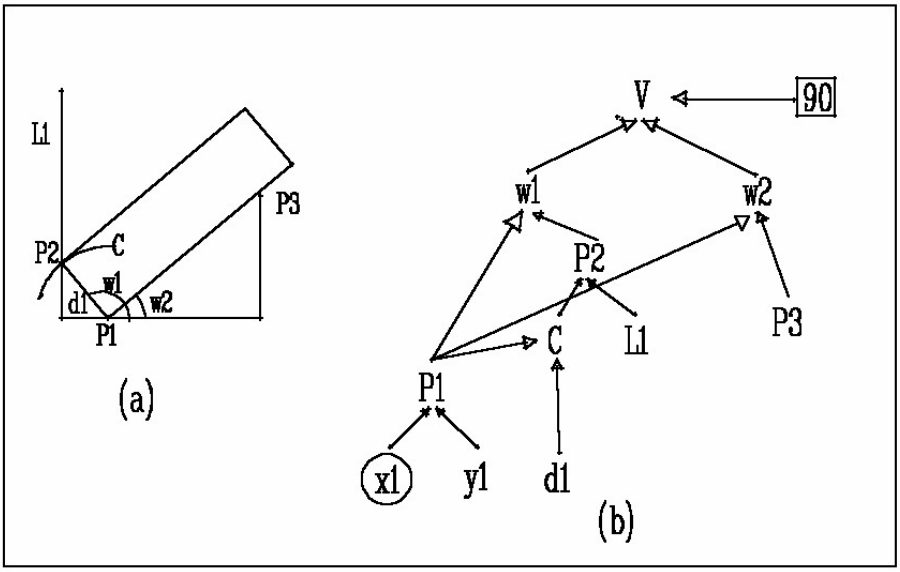


Abb. 6

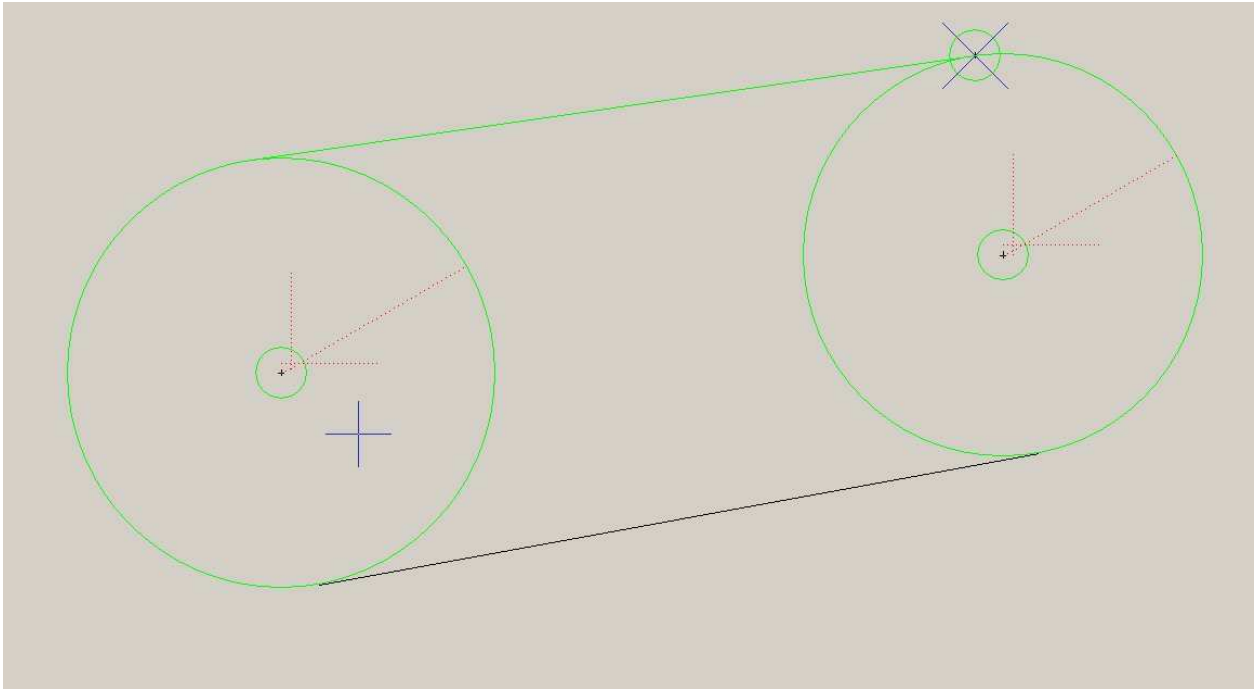
Im ersten Beispiel sind zwei Dreiecke P1, P2, P3 und P4, P5, P6 gegeben. Das kleinere Dreieck soll so in das größere platziert werden, dass die Entfernungen (P1, P4), (P2, P5) und (P3, P6) gegebene Werte annehmen. Man kann sich vorstellen, dass die Dreiecke durch Schnüre verbunden sind, die stramm gezogen werden sollen. Sequentiell erzeugbar ist nur eine Konstruktion, wo die Punkte P4, P5 und P6 als Polarpunkte zu den Punkten P1, P2, P3 erzeugt werden mit vorgegebenen Winkeln w1, w2 und w3. Die Entfernungen (P4, P5),(P5, P6), (P6, P4) haben dann

nicht die richtigen Werte. Die Winkel müssen dann so variiert werden, dass die Entfernungen die gewünschten Werte haben.

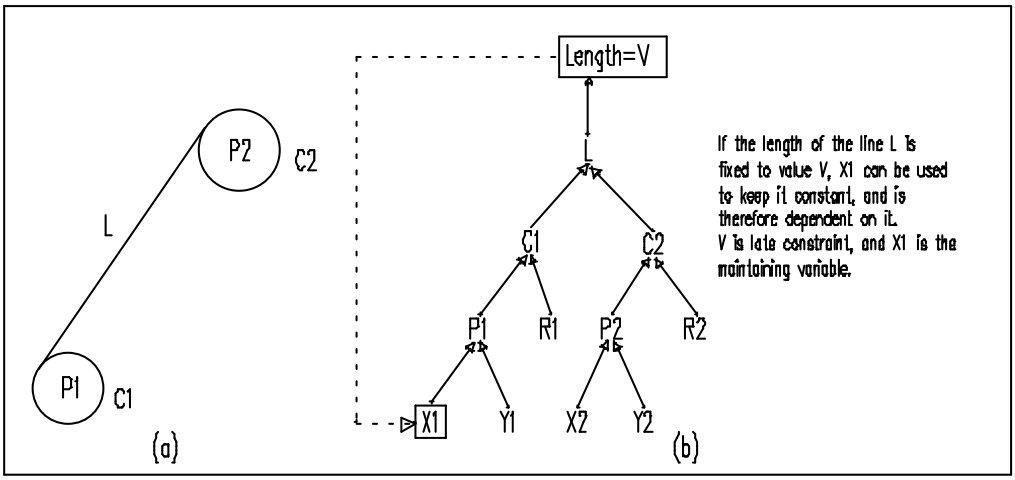
Im zweiten Beispiel ist sei eine Wanne gegeben. In diese Wanne soll ein Klotz mit einer gegebenen Breite d1 gelegt werden. Für die Berechnung von Punkt P1 benötigt man Punkt P2 und umgekehrt. Hier ist sequentiell eine Konstruktion möglich, wo P1 an einer Stelle des Bodens der Wanne gelegt wird. Dann kann P2 berechnet werden über einen Kreis um P1 mit dem Radius d1. Der sich ergebende Winkel P2, P1, P3 ist jedoch nicht 90°. Die X-Koordinate von P1 muss so variiert werden, dass der echte Winkel erreicht wird.

Um dies zu erreichen, werden in dem DAG nachträglich zusätzliche Kanten von einem inneren Knoten zu einem Blatt einfügt, die dann Zyklen erzeugen. Die Berechnung erfolgt dann, durch eine Iteration. Die Werte in den Blättern werden solange variiert, bis in dem inneren Knoten, die gewünschten Werte erreicht werden. Diese Methode wird auch benutzt um in einem DAG bei einem inneren Knoten die Werte zu ändern.

Beispiel:



Es soll der Tangentialpunkt oben links auf eine vorgegebene Koordinate gebracht werden. Dieser Punkt ist abhängig von den Mittelpunkten und den Radien der beiden Kreise. Von diesen 6 Werten müssen 2 ausgewählt werden. Im Graphen wird eine Rückkante von dem Tangentialpunkten zu den beiden Blättern erzeugt.



Hier ist die Linie L als Tangente zwischen den beiden Kreisen C1 und C2 definiert. Soll nun die Länge von L vorgegeben werden, so kann dies z.B. durch eine geeignete X Koordinate von Punkt P1 erfüllt werden. Daher wird eine Constraint Kante von dem Knoten der die Länge von L angibt zu dem Blatt mit der X Koordinate von P1 gezogen.

Diese Zyklen im Graphen können nicht durch schrittweise Evaluierung gelöst werden. Sie müssen jeweils als ganzes durch ein Gleichungssystem gelöst werden.

Zu Lösung dieses Gleichungssystem kann ein iterative Methode das **Newton-Raphson-Verfahren** benutzt werden. Hat man nur eine oder 2 Gleichungen kann man auch eine **Intervallschachtelung (regula falsi)** benutzen.

So lässt sich ein Punkt, der abhängig von Variablen ist verschieben. Aus den Variablen, die diesen Punkt beeinflussen werden 2 ausgewählt. Für diese beiden Variablen x und y wird jeweils ein Intervall <x1...x2> bzw. <y1...y2> ausgewählt. Dies wird ausgehend von den aktuellen Werten für x und y gebildet, z.B. plus und minus 20%. Mit der 2 dimensionalen Variante der Intervallschachtelung wird eine Lösung ermittelt. Soll ein Punkt um eine größere Distanz verschoben werden, kann man dies schrittweise machen. Für jeden Schritt wird die Lösung ermittelt. Die neuen Zielwerte sind dann näher an den alten Werten, damit ist auch die Veränderung bei den Variablen geringer und die Lösung liegt eher in dem Intervall.

Skiziertechnik

Der Aufbau eines solchen parametrischen Modells ist recht aufwendig und muss gut geplant werden um sequentiell zu der gewünschten Konstruktion zu kommen. Dies kann durch eine Eingabe mit Hilfe der **Skiziertechnik** beschleunigt werden.

Beispiel: Eingabe eines Polygons

Es wird zunächst die grobe Kontur eingegeben. Dabei werden automatisch folgende Anpassungen vorgenommen:

- Linien die fast waagrecht oder senkrecht sind werden waagrecht bzw. senkrecht ausgerichtet.
- Bei jeder X- oder Y-Koordinate wird überprüft, ob sie fast gleich mit einer bereits vorhandenen X oder Y Koordinate ist. Wenn ja wird die Koordinate gleichgesetzt.

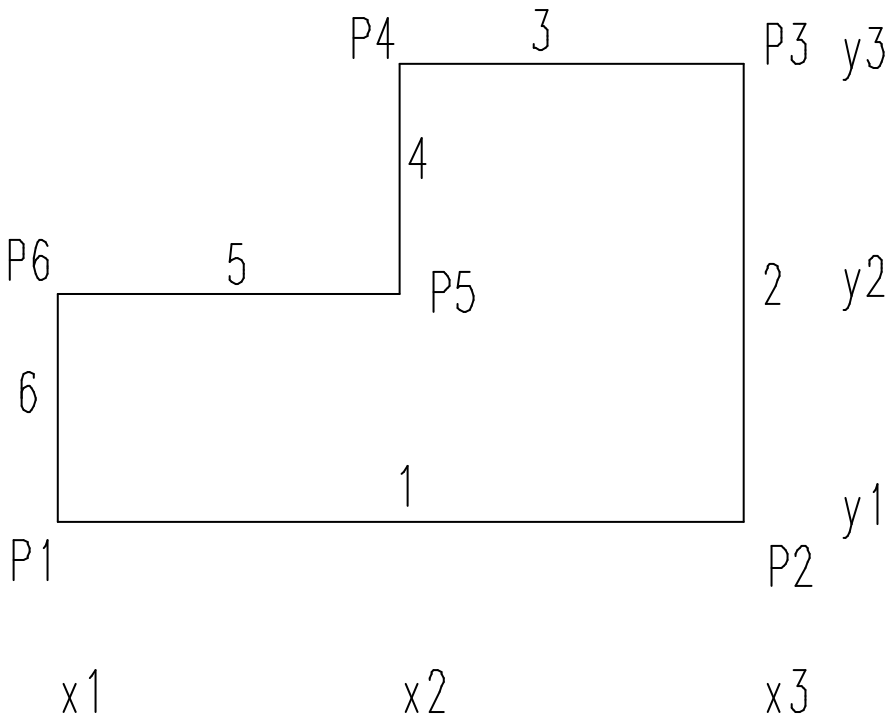
Am Ende hat man folgendes Polygon:

```
lines:array[1..nmax]of record
  l:pdb;
  p1,p2:integer;{index in pxx,pyy des punktes}
 end;
```

`p1 und p2` zeigen auf ein Punktfeld:

```
pxx,pyy:array[1..npmax]of integer;
//Index der X bzw. Y Koordinate in xxx bzw. yyy

xxx,yyy:array[1..nmax]of tx;
// Feld der vorkommenden X und Y Koordinaten
```



Linie	von	nach
1	1	2
2	2	3
3	3	4
4	4	5
5	5	6
6	6	1

Punkt	X	Y
1	x1	y1
2	x3	y1
3	x3	y3
4	x2	y3
5	x2	y2
6	x1	y2

Für die einzelnen Linien kann der Benutzer Werte eingeben. Z.B.

```
Linie 1=100 => x3-x1=100
Linie 3= 40 => x3-x2= 40
```

Daraufhin ermittelt das System automatisch x2-x1=60 und damit ist Linie 5 bestimmt. Analog kann bei den senkrechten Linien verfahren werden.

Um diese Abhängigkeiten festzustellen wird eine Dreiecksmatrix benutzt:

```
ax: array[1..nmax,1..nmax]of tx;

ax[d,i]=x[i+d]-x[i]
```

Analog für die Y Werte:
`ay[d,i]=y[i+d]-y[i]`

Die beiden Matrizen werden mit den negativen Abständen vor besetzt.

```
for i:=1 to nx-1 do for j:=1 to nx-i do begin
 ax[i,j]:=-(xxx[j+i]-xxx[j]);
end;
for i:=1 to ny-1 do for j:=1 to ny-i do begin
 ay[i,j]:=-(yyy[j+i]-yyy[j]);
end;
```

Hat der Benutzer für eine waagrechte Linie beginnend bei X-Koordinate in `xxx[p1x]` mit der bisherigen Länge `dx` einen Wert `dd` eingegeben wird `recompute(ax,nx,dx,p1x,real2tx(dd))` aufgerufen. Diese Prozedur ändert die Abstände, die durch diese Eingabe ebenfalls festgelegt werden.
Für die Matrix ax gilt:

```
(A) ax[d,p]:=ax[d-i,p]+ax[i,p+d-i]
```

Dabei gilt `i<d.` Die Formel besagt:
Abstand zwischen Punkt p und Punkt `p+d`=
Summe der Abstände zwischen p und `p+d-i` und zwischen `p+d-i` und `p+d`

Ist also `ax[d-i,p]>0` und `ax[i,p+d-i]>0,` und damit bestimmt, so kann `a[d,p]` berechnet werden.

Ein Abstand kann aber nicht nur die Summe von 2 Abständen sein, sondern auch die Differenz. Daraus ergibt sich:

```
(B) ax[d,p]:=ax[d+i,p]-ax[i,p+d]
```
mit `d+i+p<=nx` und daraus `i<= nx-d-p`

```
(C) ax[d,p]:=ax[d+i,p-i]-ax[i,p-i]
mit i<p,
d+i+p-i<=nx und daraus d+p<=nx
i+p-i<=nx   und daraus p<=nx
```

Bei jeder Eingabe wird der entsprechende Abstand positiv eingetragen und entsprechend Formeln A, B, C weitere Werte positiv gesetzt. Die Koordinaten der Punkte werden dann neu berechnet und Linien zwischen Punkten, deren Abstände festliegen werden gekennzeichnet.

Der Benutzer kann so für die horizontalen und vertikalen Abstände solange Werte eingeben, bis alle Abstände festliegen.

Variantenprogrammierung

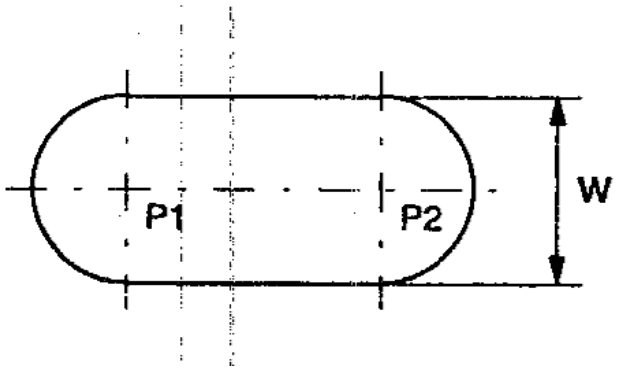
Eine weitere Methode Varianten zu definieren ist die Definition durch ein Programm. Die einfachste Form ist, dass dieses Programm in einer Programmiersprache als Programm in Textform erstellt wird. Viele CAD Systeme haben für die Variantenprogrammierung spezielle Skript- bzw. Makrosprachen, z.B. Auto-LISP bei AutoCAD.

Eigenschaften der Makrosprachen

- Kontrollstrukturen ähnlich wie Programmiersprachen
- Variablen
- Prozeduren und ihre Verschachtelung
- Zugriff auf die Geometrieelemente
- Erzeugung von Geometrieelementen
- Zugriff auf Systemparameter und auch Änderung derselben
- Schrittweise Ablaufverfolgung (Trace)
- Fehlerbehandlung zur Laufzeit
- Einbindung externer Programme
- Schreiben und Lesen von Daten

Das Programm wird, wie bei Skriptsprachen üblich, i.A. in Textform vorliegen. Zur Ablaufbeschleunigung und evtl. auch um das Know-How zu schützen, kann auch ein Compiler vorgesehen sein, der das Programm in eine Binärform übersetzt.

Beispiel (HP-ME10):



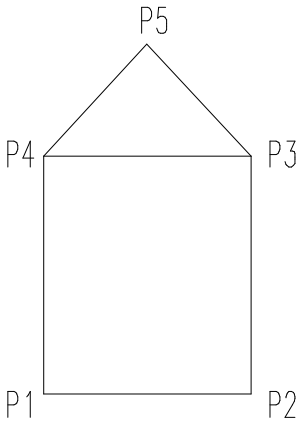

```
DEFINE Nut
    LOCAL W
    LOCAL P1
    LOCAL P2
    LOCAL V
    READ NUMBER "Bitte Nutbreite eingeben" W
    LOOP
        FOLLOW OFF
        COLOR WHITE
        LINETYPE SOLID
        READ PNT "Ersten Mittelpunkt eingeben" P1
        READ PNT "Zweiten Mittelpunkt eingeben" P2
        LET V (NORMAL (P2 - P1) * (W / 2) )
        ARC CEN_BEG_END P1 (P1 + V) (P1 - V)
        ARC CEN_BEG_END P2 (P2 - V) (P2 + V)
        LINE POLYGON (P1 - V) (P2 - V)
        LINE POLYGON (P1 + V) (P2 + V)
        LET V (V * 1.2)
        COLOR YELLOW
        LINETYPE DOT_CENTER
        LINE POLYGON (P1 - V) (P1 + V)
        LINE POLYGON (P2 - V) (P2 + V)
        LET V (ROT V90)
        LINE POLYGON (P1 + V) (P2 - V)
    END_LOOP
END_DEFINE
```

Erstellung von Varianten ohne Programmierung

Im einfachsten fall geschieht dies durch die so genannte **sequentielle Rekonstruktion**. Dabei werden die Bedienschritte zur Erstellung der Geometrie

aufgezeichnet. Beim Abspielen wird für jede Punktbestimmung bzw. Objektauswahl ein interaktiver Dialog eingefügt.

Beispiel:

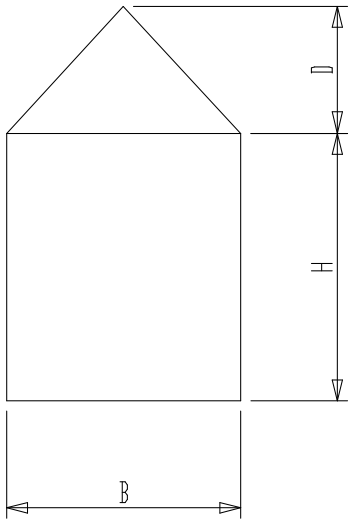


(1) Rechteck P1(Eingabe),P3(Eingabe)
(2) Linie Startpunkt: P4 Endpunkt X = Mitte (P4,P3), Y= P5.y(Eingabe)
(3) Linie P5,P3

Die unterstrichenen Teile werden eingegeben und können variiert werden.

Ableitung aus einem Muster mit assoziativer Bemaßung (VarioCAD)

Es wird ein Segment erzeugt, das ein Muster des zu erzeugenden Elementes incl. einer symbolischen assoziativen Bemaßung enthält.



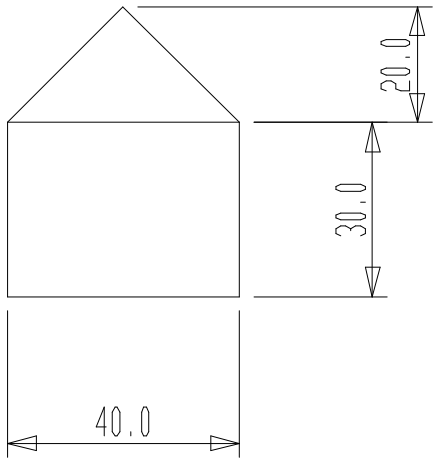
Bei der Erzeugung des Makros gibt man für alle vorkommenden X und Y Koordinaten eine Formel ein. Im obigen Beispiel wäre dies:

```
X1=0; X2= B/2; X3=B
Y1=0; Y1=H; Y3=H+B
```

Bei der Erzeugung einer Variante werden alle vorkommenden Variablen, hier B,H und D abgefragt. Gibt man ein:

```
B=40; H=30; D=20
```

so wird folgende Variante erzeugt:



In der Variantendefinition sind für alle Elemente die Formeln für Anfangs- und Endpunkt abgespeichert.

```
// 1. Dachlinie
   6:  E  1   0.000000  63.444480  27.756960  93.444480 ["]
0,H,B/2,H+D,
// 2.Dachlinie
   8:  E  1  27.756960  93.444480  55.513920  63.444480 ["]
B/2,H+D,B,H,
// Rechteck mit 4 Linien
  17:  M 73 18 0 0 0 1 35 0 1 1 44 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  18:   E  1   0.000000   0.000000  55.513920   0.000000 ["]
0,H,B,H,
  20:   E  1  55.513920   0.000000  55.513920 -63.444480 ["]
B,H,B,0,
  22:   E  1  55.513920 -63.444480   0.000000 -63.444480 ["]
B,0,0,0,
  24:   E  1   0.000000 -63.444480   0.000000   0.000000 ["]
0,0,0,H,
```