Fachhochschule Nordwestschweiz
Hochschule für Technik

Algoria

MSE P7b – Symbol-Morphing

September 2011

Autor: Raphael Schweizer

Advisor: Prof. Dr. Christoph Stamm
Institut für Mobile und Verteilte Systeme FHNW

Fachhochschule Nordwestschweiz
Hochschule für Technik

Raphael Schweizer
Hinterer Hafen 348
5224 Unterbözberg

+41 (0)79 378 42 33
fhnw@schweizer-informatik.ch

Satz: XeLaTeX, Grafiken: TikZ

Algoria ist eine Anwendung zur Skizzenerkennung auf Convertibles. Es können typische Datenstrukturen aus der Informatik - wie Arrays, Listen und Bäume - erkannt werden. Auf den „zum Leben erweckten“ Strukturen können anschliessend häufige Operationen (z.B. Einfügen, Löschen, Suchen, Sortieren, etc.) animiert ausgeführt werden.

Diese Arbeit beschreibt die Implementation eines Symbol-Morphers, einer Komponente, die ein skizziertes und erkanntes Symbol so verändert, dass es dem zu erkennenden Symbol besser entspricht, ohne seine „Handschrift“ zu verlieren.

Das Projekt Algoria ist *Work in Progress* in experimentellem Stadium. Die vorliegende Arbeit entspricht mindestens der Revision 3364 des Codes auf dem zur Entwicklung von Algoria verwendeten Versionierungsserver.

Inhaltsverzeichnis

1 Einleitung

Algoria erlaubt Dozierenden, während dem Zeichnen von Datenstrukturen der Klasse zugewandt zu bleiben und komplizierte Sachverhalte mittels Annotationen und Animationen verständlich darzustellen. Algoria erkennt skizzierte Arrays, Listen und Bäume und bildet diese strukturiert nach, so dass Algorithmen animiert ausgeführt werden können.

1.1 Ziel

Die erkannten Symbole müssen einerseits klar von der Skizze unterscheidbar sein, andererseits soll der Bruch nicht zu stark sein, so dass der Benutzer auf natürliche Art und Weise weiter mit „seiner“ Datenstruktur interagieren kann. In einem (zukünftig denkbaren) Mehrbenutzer-Algoria können so zudem die erkannten Elemente einfacher dem jeweiligen Urheber zugeordnet werden.

Das Ziel der vorliegenden Arbeit ist die Implementation eines Symbol-Morphers, einer Komponente, die aus skizziertem und erkanntem Symbol eine möglichst passende und gleichzeitig möglichst wenig vom skizzierten Symbol abweichende Repräsentation generiert. Um die Abweichung vom skizzierten Symbol zu messen, soll eine geeignete Metrik gefunden werden. Die Passgenauigkeit definiert sich über die Constrainterfüllung.

2 Morpher

Der Symbol-Morpher generiert aus einem gegebenen erkannten Symbol eine veränderte Repräsentation, die dem ursprünglichen Symbol ähnlich sieht und die Constraints besser erfüllt.

2.1 Ähnlichkeitsmetrik

Als Ähnlichkeitsmetrik s wird die Summe der Abstände im Quadrat der ursprünglichen Punkte P zu den entsprechenden Punkten des optimierten Symbols gewählt. (Formel 2.1) Diese Metrik stellt offensichtlich sicher, dass das optimierte Symbol nahe am ursprünglich gezeichneten ist, da die einzigen Variablen der Optimierung eben diese Punkte sind. Gleichzeitig ist die Implemetierung und Berechnung einfach und nachvollziehbar.

$$s = \sum_{p_i \in P} \left| \overline{p_{i_{Skizze}} p_{i_{Optimiert}}} \right|^2 \tag{2.1}$$

2.2 Optimierung

In einem ersten Anlauf wurde versucht, die Constraints zu linearisieren, also z.B. Winkelfunktionen durch Taylor- bzw. Maclaurin-Reihen hinreichender Genauigkeit anzunähern. Der Vorteil einer rein linearen Programmierung (v.a. schnellere, deterministische Solver) wägt den grösseren Aufwand bei der Programmierung und die anschliessend schlechtere Les- und Wartbarkeit des Codes (zumindest für Nicht-Mathematiker) aber nicht auf. Ausserdem ist die Konvexität des Lösungsraumes nicht per se gegeben. Die Verwendung eines generischen nicht-linearen Optimierers (s. Abschnitt 3.3) scheint angebracht. Ein weiterer Vorteil ist, dass die für die Constraint-Berechnung verwendeten Formeln grösstenteils für die Optimierung wiederverwendet werden können.

Mit einem allgemeinen, nicht-linearen Minimierungs-Modell kann somit z.B. die Optimierung des `angle`-Constraints für drei Punkte a, b, c und dem Ziel-Innenwinkel α_t wie in Formel 2.2 beschrieben werden.

$$\min \left\{ \left| \alpha_t - \arccos \left(\frac{\overline{ab} \cdot \overline{cb}}{\left\| \overline{ab} \right\|_2 \cdot \left\| \overline{cb} \right\|_2} \right) \right| \right\} \tag{2.2}$$

Solche und ähnliche Terme können auch von Laien formuliert werden um neue, optimierbare Constraints zu programmieren. Implementationsspezifische Details werden in Abschnitt 3 erläutert.

Der zweite Teil der Optimierung besteht nun darin, die Terme unter Berücksichtigung des Ähnlichkeitsmasses zu minimieren und die Koeffizienten für die ursprünglichen Punkte zu übernehmen, um so das Ursprungssymbol in seine neue Repräsentation zu transformieren.

3 Implementation

Als Optimierungs-Framework wird *Microsoft Solver Foundation*[1] verwendet. Diese Entscheidung wurde aufgrund der in Abschnitt 3.3 aufgeführten Eigenschaften getroffen.

3.1 Constraints

Ein optimierbarer Constraint muss das Interface `IOptimizable` (Listing 3.1) implementieren. Zur einfachen Konstruktion häufiger Terme steht die Klasse `OptimizableHelper` zur Verfügung, die z.B. Methoden für Längen, Distanzen und Skalarprodukte enthält.

```
1 public interface IOptimizable : IConstraint
2 {
3   Term Minimization(IConstraintDescription description, Decision x, Decision y);
4   IEnumerable<string> UsedPoints(IConstraintDescription description);
5   bool HasNonNativePoints { get; }
6 }
```

Listing 3.1: IOptimizable Interface

Aus dem Testset in Abschnitt 4 ergeben sich die ersten sieben Constraints in Tabelle 3.1. Die vier restlichen Constraints sind für das Testset nicht relevant, ihre Implementation ergibt sich trivialerweise aus vorhergehenden Constraints.

Constraint	Argumente
coincident	(Punkt, Punkt)
equalLength	(Linie, Linie) bzw. (Punkt, Punkt, Punkt, Punkt)
parallel	(Linie, Linie)
perpendicular	(Linie, Linie)
angle	(Punkt, Punkt, Punkt) bzw. (Punkt, Punkt, Punkt, Punkt)
sameSize	(Kreis, Kreis)
liesOn	(Punkt, Kreis)
horizontal	(Linie)
vertical	(Linie)
sameX	(Punkt, Punkt)
sameY	(Punkt, Punkt)

Tabelle 3.1: Optimierbare Constraints

Die Implementation eines optimierbaren Constraints soll anhand des `angle`-Constraints beispielhaft untersucht werden. In Listing 3.2 ist ein Auszug des für die Implementation von `IOptimizable` relevanten Codes zu finden. Der Klassenname `angle4` weist auf die Argumentliste (vier Punkte) hin, der `angle`-Constraint für drei Punkte heisst entsprechend `angle3`. Die Überladung erfolgt mittels `Name` und `ComponentTypes`. Der zu minimierende Term entspricht bis auf einen Normalisierungsfaktor der Formel 2.2. Dieser Faktor gibt

[1] `http://www.solverfoundation.com/`

dem Term das nötige Gewicht um bei der Optimierung adäquat berücksichtigt zu werden. Mit den für diese Arbeit implementierten Constraints müssen keine weiteren Normalisierungen vorgenommen werden. Es ist allerdings möglich, dass mit einer viel grösseren Constraint-Menge ein konsistentes Normalisierungs-Modell angewandt werden muss – ähnlich denjenigem für die Constraint-Auswertung im `Symbolizer` – damit durch einzelne, extreme Terme kein Ungleichgewicht entsteht.

Für die Optimierung des `angle`-Constraints werden die vier Punkte (`UsedPoints`, `HasNonNativePoints`) aus der LADDER-Datei benutzt, der `parallel`-Constraint beispielsweise nutzt die vier Punkte der in der LADDER-Datei genannten zwei Linien (Listing 3.3), der `liesOn`-Constraint (Listing 3.4) wiederum benötigt einen Punkt sowie den Mittelpunkt und den Radius des Kreises. Mittelpunkt und Radius sind nach Festlegung des Kreises aber eindeutig bestimmt (im Gegensatz zu den zwei Linien-Endpunkten), so dass `HasNonNativePoints` – das angibt, ob weitere Punkte als die in der LADDER-Datei verwendeten, benutzt werden – `false` zurück geben kann.

```
1  public class Angle4 : IOptimizable
2  {
3    public string Name { get { return ”angle”; } }
4    public Type[] ComponentTypes {
5      get {
6        return new[] {
7          typeof(PointPart), typeof(PointPart), typeof(PointPart), typeof(PointPart)
8        };
9      }
10   }

12   // Weitere IConstraint-spezifische Implementationsdetails [...]

14   public Term Minimization(IConstraintDescription description, Decision x, Decision y)
15   {
16     var a = description.ShapeParameters[0];
17     var b = description.ShapeParameters[1];
18     var c = description.ShapeParameters[2];
19     var d = description.ShapeParameters[3];
20     var dotProd = OptimizableHelper.DotProduct(a, b, d, c, x, y);
21     var lengthProd = OptimizableHelper.Distance(a, b, x, y) *
22       OptimizableHelper.Distance(d, c, x, y);
23     var targetAngle = Math.PI * (int)description.Parameters[0];

25     return Model.Power(targetAngle - Model.ArcCos(dotProd / lengthProd) * 180, 2);
26   }

28   public IEnumerable<string> UsedPoints(IConstraintDescription description)
29   { return description.ShapeParameters; }

31   public bool HasNonNativePoints { get { return false; } }
32 }
```

Listing 3.2: Auszug aus der Implementation des `angle`-Constraints

```
1  public IEnumerable<string> UsedPoints(IConstraintDescription description)
2  { return OptimizableHelper.LinesToPoints(description.ShapeParameters); }

4  public bool HasNonNativePoints { get { return true; } }
```

Listing 3.3: Auszug aus der Implementation des `parallel`-Constraints

```
1  public IEnumerable<string> UsedPoints(IConstraintDescription description)
2  {
3    var p = description.ShapeParameters[0];
4    var c = description.ShapeParameters[1];
5    return new[] {p, c + ”.Center”, c + ”.Radius”};
6  }

8  public bool HasNonNativePoints { get { return false; } }
```

Listing 3.4: Auszug aus der Implementation des `liesOn`-Constraints

3.2 Symbole

Nach der Minimierung der Constraint-Terme durch die Klasse `IMVS.Algoria.Symbols.SolverFoundation` werden die ermittelten Werte auf die verwendeten `PointParts` übertragen. Die Symbole als Observer ihrer Punkte aktualisieren anschliessend die Repräsentationen. Dazu wurde die Klassenhierarchie des `Componizer`- und `Symbolizer`-Namespaces stark überarbeitet. Die `IComponent`-Objekte, die nur eine kurze Lebenszeit zwischen Erkennung von Basis-Komponenten (Linien, Ellipsen, Kreise, Kreisbögen und Pfade) und Erstellung von entsprechenden Symbolen hatten, wurden ersatzlos gestrichen. Als Repräsentation der Symbole werden neu .NET Geometrien[2] verwendet. Diese können einfach transformiert und kombiniert werden, was vor allem für die spätere Erzeugung von zusammengesetzten Symbolen von Bedeutung ist.

Für die Linie wird der gezeichnete, unterabgetastete `Stroke` benutzt. Mit affinen Transformationen Rotieren, Skalieren und Verschieben (Listing 3.5) wird diese den optimierten Werten der Endpunkte angepasst. Für den Kreis wird eine `EllipseGeometry`, deren Radius und Mittelpunkt angepasst wird, verwendet. (Abbildung 3.1)

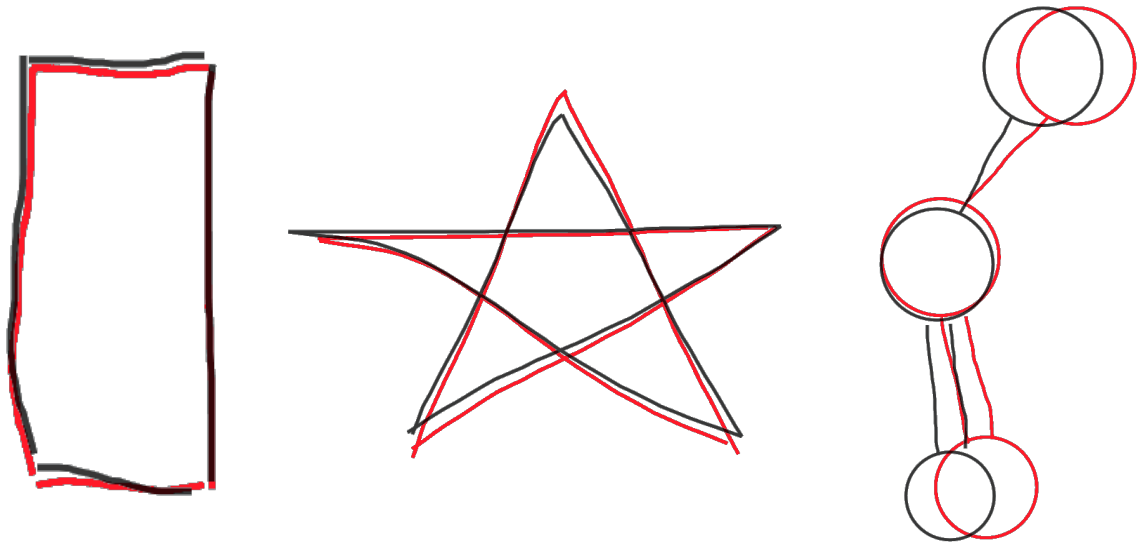


Abbildung 3.1: Ursprünglich gezeichnete (schwarz) und optimierte (rot) Symbole

```
1  var matrix = Matrix.Identity;

3  var angleDiff = Angle - Math.Atan2(dOldY, dOldX);
4  matrix.RotateAt(angleDiff * Rad2Deg, midX, midY);

6  var lengthQuot = Math.Sqrt(dNewX * dNewX + dNewY * dNewY) /
7    Math.Sqrt(dOldX * dOldX + dOldY * dOldY);
8  matrix.ScaleAt(lengthQuot, lengthQuot, midX, midY);

10 var origin = matrix.Transform(op1);
11 matrix.Translate(P1.X - origin.X, P1.Y - origin.Y);
```

Listing 3.5: Berechnung der affinen Transformation für die Linien-Anpassung

[2] `http://msdn.microsoft.com/en-us/library/ms751808.aspx`

3.3 Solver

Die unmittelbare Optimierung nach dem Zeichnen eines Symbols bedingt eine möglichst kurze Latenzzeit des Solvers. Wichtig sind auch eine einfache Anbindung an .NET, liberale Lizenzbedingungen und ein gutes Geschwindigkeits-Genauigkeits-Verhältnis. Diese Kriterien haben den klaren Ausschlag zugunsten der *Microsoft Solver Foundation* gegeben.

3.3.1 Microsoft Solver Foundation

Microsoft Solver Foundation (MSSF) ist ein Framework zur Anbindung unterschiedlichster Solver an die .NET-Platform und *Microsoft Excel*. Es sind drei verschiedene Versionen erhältlich: die kostenfreie Express-Version enthält Problemgrössenbeschränkungen und kein Recht zur Weiterverteilung, muss also vom Endbenutzer separat heruntergeladen und installiert werden. Die Standard-Version entspricht bis auf die Verteilung der Express-Version. Die Enterprise-Version schliesslich muss über *Gurobi*[3] lizenziert werden und enthält unter anderen den Gurobi-Solver, hebt die Problemgrössenbeschränkung auf und bietet Unterstützung für Multi-Core-Systeme. Für Algoria ist die Express-Version vorerst ausreichend, zur Verteilung im akademischen Umfeld kann eine entsprechende Version über MSDNAA bezogen werden.

Die zu Beginn der Arbeit verfügbare Version 3.0 stellte als Solver für nicht-lineare Optimierung den *Hybrid Local Search Solver* (HLS) zur Verfügung. Dieser versucht mittels randomisiertem Hill-Climbing eine vorgegebene Lösung zu verbessern. Bei stark eingeschränkten Lösungsräumen versagt er aber oft und verschlechtert unter Umständen sogar die Initiallösung.[4] Die kurz vor Ende der Arbeit erschienene Version 3.1 ist durch einen Nelder-Mead Solver (NM) ergänzt worden.[5] Dieser benutzt ein Downhill-Simplex-Verfahren,[6] das nicht wesentlich schneller aber viel stabiler als der HLS ist.

Die Anbindung erfolgt intuitiv und solverunabhängig über .NET-Klassen (s. Listing 3.2). Mit der Version 3.1 sind auch Sprachneuerungen von C# 4.0 hinzu gekommen, die v.a. den Zugriff und das Binding auf Parameter bzw. von Daten mit Lambda-Expressions anstelle von `string`s vereinfachen.

3.3.2 MATLAB

MATLAB[7] kann über eine C-API, DDE oder COM angesprochen werden. Allen Zugriffswegen gemeinsam ist eine – verglichen mit der MSSF – grosse Latenzzeit, die der Umsetzung von .NET nach nativen Datentypen geschuldet ist. Für relativ kleine Modelle, wie sie bei Algoria vorliegen, lohnt sich dieser Aufwand nicht. Der .NET-Wrapper und der .NET-Compiler, der MATLAB-Code nach .NET übersetzt, konnte mangels Lizenz[8] nicht getestet werden.

[3] `http://www.gurobi.com`
[4] `http://social.msdn.microsoft.com/Forums/en-US/solverfoundation/thread/3833f4b1-a598-4f16-9cb9-604324de0810`
[5] `http://social.msdn.microsoft.com/Forums/en-US/solverfoundation/thread/5fa815e0-98cd-469e-babf-e786701263ab`
[6] Stark vereinfacht: in das Bild der zu optimierenden Funktion wird ein N+1-dimensionaler Simplex gelegt, der Richtung einem lokalen Optimum bewegt wird, indem in jedem Schritt der ‚schlechteste' Punkt durch einen ‚besseren' ausgetauscht wird. (N = Anzahl Parameter)
[7] http://www.mathworks.ch/products/matlab/index.html
[8] s.a. Mail „Verwendung Matlab/Simulink in Projekten mit Industriepartner" der *Direktion der Hochschule für Technik* vom 09.07.2010 zur Aufkündigung der Kooperation mit *MathWorks*

Die Ansteuerung von MATLAB wirkt in einer reinen .NET-Umgebung wie ein Fremdkörper. Hilfsklassen für eine ähnlich bequeme Bedienung wie MSSF müssten erst erstellt werden. Gleichwohl hat eine testweise MATLAB-Implementierung numerisch gute Resultate gezeigt. Der hierfür benötigte Code ist in der Versions-Historie auf dem Entwicklungsserver festgehalten.

3.4 Backlog

Die Umstellung der `ISymbol`-Klassen erfolgte unabhängig von der Darstellung. Dabei traten Layout-Probleme im Zusammenhang mit dem Datenfluss von Komponente über Symbol zu Repräsentation im UI auf, die noch offen sind. Die Symbol-Optimierung kann deshalb momentan ausschliesslich im „Snapshot"-Modus verwendet werden.

Eine Repräsentation, die die skizzierten `Strokes` verwendet ist nur für die Linie vorhanden. Kreise, Kreisbögen und Ellipsen verwenden ‚synthetische' Geometrien, die auch nur im Fall des Kreises optimierbar ist.

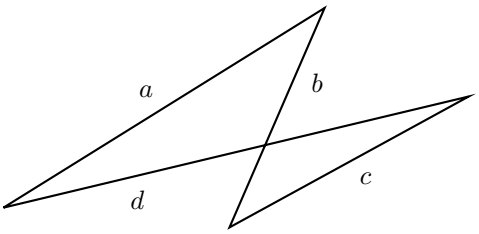
3.5 Fazit

Die vorliegende Arbeit ist der Grundstein für ein stabiles, erweiterbares Symbol-Morphing, dessen Funktionstüchtigkeit anhand eines Testsets mit Anspruch auf Verallgemeinerbarkeit unter Beweis gestellt wurde. Unvorhergesehene Probleme bei der Integration haben eine durchgehend implementierte Lösung verhindert, was dem AHA-Effekt für die bereits optimierbaren Symbole keinen Abbruch tut. Es ist absehbar, dass die über den Umfang dieser Arbeit hinausgehenden Teile im Rahmen der weiteren Tätigkeit des Autors an Algoria in Kürze folgen werden.

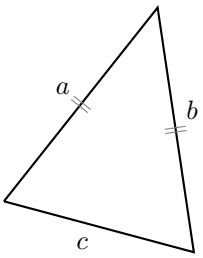
4 LADDER-Testset

4.1 Basisgeometrien

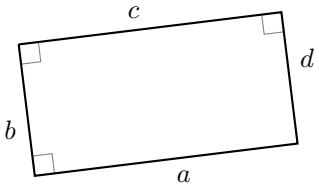
```
(define shape Quadrilateral
  (description "Is it bi-bilateral?")
  (components
    (Line a)
    (Line b)
    (Line c)
    (Line d)
  )
  (constraints
    (coincident a.P1 c.P2 70)
    (coincident a.P2 d.P2 70)
    (coincident b.P1 c.P1 70)
    (coincident b.P2 d.P1 70)
  )
)
```




```
(define shape IsoscelesTriangle
  (description "Greek: iso = same, skelos = leg")
  (components
    (Line a)
    (Line b)
    (Line c)
  )
  (constraints
    (equalLength a b 80)
    (coincident a.P1 b.P1 50)
    (coincident a.P2 c.P1 70)
    (coincident b.P2 c.P2 70)
  )
)
```

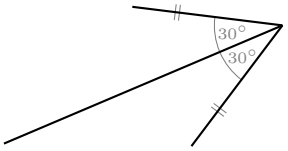



```
(define shape Rectangle
  (description "A very disciplined quadrilateral")
  (components
    (Line a)
    (Line b)
    (Line c)
    (Line d)
  )
  (constraints
    (coincident a.P2 b.P1 30)
    (coincident b.P2 c.P1 30)
    (coincident c.P2 d.P1 30)
    (coincident d.P2 a.P1 30)

    (perpendicular a b 20)
    (perpendicular b c 20)
    (perpendicular c d 20)
  )
)
```

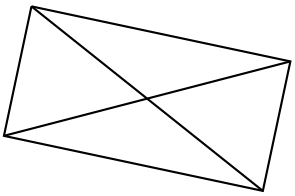


4.2 Zusammengesetzte Geometrien

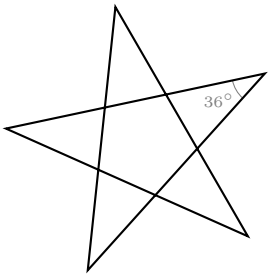
```
(define shape Arrow
  (description "A simple arrow")
  (components
    (Line shaft)
    (Line head1)
    (Line head2)
  )
  (constraints
    (longer shaft head1 10)
    (coincident shaft.P2 head2.P1 20)
    (coincident shaft.P2 head1.P1 20)
    (angle 30 head1.P2 head1.P1 shaft.P2 shaft.P1 30)
    (angle 30 shaft.P1 shaft.P2 head2.P1 head2.P2 30)
    (equalLength head1 head2 40)
  )
)
```



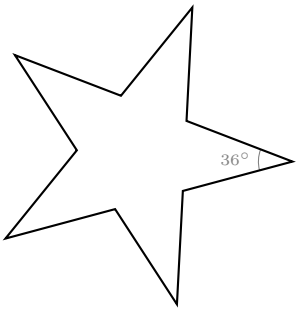

```
(define shape Checkbox
  (description "A captured X")
  (components
    (Rectangle r)
    (Line u)
    (Line v)
  )
  (constraints
    (coincident r.a.P1 u.P1 20)
    (coincident r.a.P2 v.P1 20)
    (coincident r.c.P1 u.P2 20)
    (coincident r.c.P2 v.P2 20)
  )
)
```



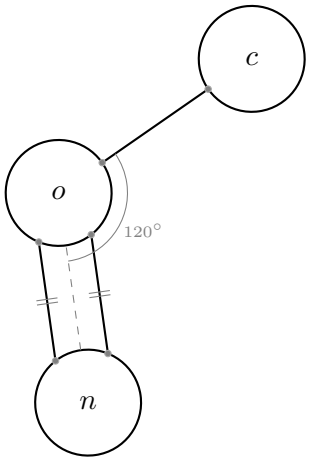
```
(define shape Pentagram
  (description "orientation independent")
  (components
    (Line a)
    (Line b)
    (Line c)
    (Line d)
    (Line e)
  )
  (constraints
    (equalLength a b 40)
    (equalLength a c 40)
    (equalLength a d 40)
    (equalLength a e 40)

    (coincident a.P2 b.P1 30)
    (coincident b.P2 c.P1 30)
    (coincident c.P2 d.P1 30)
    (coincident d.P2 e.P1 30)
    (coincident e.P2 a.P1 30)

    (angle 36 c.P1 c.P2 d.P1 d.P2 40)
    (angle 36 b.P1 b.P2 c.P1 c.P2 40)
    (angle 36 a.P1 a.P2 b.P1 b.P2 40)
    (angle 36 d.P1 d.P2 e.P1 e.P2 40)
    (angle 36 e.P1 e.P2 a.P1 a.P2 40)
  )
)
```



```
(define shape Star
  (description "Just another starlet")
  (components
    (Line a)
    (Line b)
    (Line c)
    (Line d)
    (Line e)
    (Line f)
    (Line g)
    (Line h)
    (Line i)
    (Line k)
  )
  (constraints
    (coincident a.P2 b.P1 30)
    (coincident b.P2 c.P1 30)
    (coincident c.P2 d.P1 30)
    (coincident d.P2 e.P1 30)
    (coincident e.P2 f.P1 30)
    (coincident f.P2 g.P1 30)
    (coincident g.P2 h.P1 30)
    (coincident h.P2 i.P1 30)
    (coincident i.P2 k.P1 30)
    (coincident k.P2 a.P1 30)

    (angle 36 a.P1 a.P2 b.P1 b.P2 40)
    (angle 36 c.P1 c.P2 d.P1 d.P2 40)
    (angle 36 e.P1 e.P2 f.P1 f.P2 40)
    (angle 36 g.P1 g.P2 h.P1 h.P2 40)
    (angle 36 i.P1 i.P2 k.P1 k.P2 40)

    (equalLength a b 50)
    (equalLength a c 50)
    (equalLength a d 50)
    (equalLength a e 50)
    (equalLength a f 50)
    (equalLength a g 50)
    (equalLength a h 50)
    (equalLength a i 50)
    (equalLength a k 50)
  )
)
```




```
(define shape Molecule
  (description "An imaginary molecule")
  (components
    (Circle n)
    (Circle o)
    (Circle c)
    (Line no1)
    (Line no2)
    (Line oc)
  )
  (constraints
    (sameSize n o 80)
    (sameSize o c 80)
    (parallel no1 no2 80)
    (equalLength no2 oc 80)
    (liesOn no1.P1 n 70)
    (liesOn no2.P1 n 70)
    (liesOn no1.P2 o 70)
    (liesOn no2.P2 o 70)
    (liesOn oc.P1 o 70)
    (liesOn oc.P2 c 70)
    (angle 120 n.Center o.Center c.Center 80)
    (angle 120 no1.P1 no1.P2 oc.P1 oc.P2 90)
    (angle 120 no2.P1 no2.P2 oc.P1 oc.P2 90)
  )
)
```



Fachhochschule Nordwestschweiz
Hochschule für Technik

Ehrlichkeitserklärung

Hiermit bestätigt der unterzeichnende Autor, dass alle nicht klar gekennzeichneten Stellen von ihm selbst erarbeitet und verfasst wurden.

Unterbözberg, September 2011

Raphael Schweizer