The Juno-2
Constraint-Based
Drawing Editor

Allan Heydon
Greg Nelson
Digital Equipment Corporation
Systems Research Center (SRC)
# Issues in Constraint-Based Drawing

<table>
<thead>
<tr>
<th>Issue</th>
<th>Juno-2 Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representing constraints</td>
<td>Double-view editing</td>
</tr>
<tr>
<td>Specifying constraints</td>
<td>Entered directly with tools</td>
</tr>
<tr>
<td>Underconstrained systems</td>
<td>Hints</td>
</tr>
<tr>
<td>Redundant constraints</td>
<td>Ignore redundancy</td>
</tr>
<tr>
<td>Defining new constraints</td>
<td>Powerful extension language</td>
</tr>
<tr>
<td>Scale</td>
<td>Hierarchical structure</td>
</tr>
</tbody>
</table>

---

digital Systems Research Center
Double-View Editing

PS.MoveTo(a);
; PS.CurveTo(b,c,d);
; PS.LineTo(a);
; PS.Fill()
Constraint Solving

Syntax:

\[ \text{VAR } <\text{var}> \sim <\text{hint}> \text{ IN} \]
\[ <\text{constraint}> \rightarrow <\text{statement}> \]
\[ \text{END} \]

Example:

\[ \text{VAR } x \sim 1 \text{ IN} \]
\[ x \times x = 2 \rightarrow \text{Print}(x) \]
\[ \text{END} \]

Built-In Constraints:

\[ a \text{ HOR } b \]
\[ b \text{ VER } c \]
\[ (d, e) \text{ CONG } (e, f) \]
\[ (g, h) \text{ PARA } (i, j) \]
The REL Function

\[(x, y) \text{ REL } (a, b) = \]

the point \((x, y)\) in the coordinate system whose origin is \(a\) and whose unit \(x\) vector goes from \(a\) to \(b\).
Definitions

Predicates, Functions, Procedures:

PRED P(x) IS <constraint> END;
FUNC y = F(x) IS <constraint> END;
PROC Proc(x) IS <statement> END;

Existential Quantification:

(E <var> ~ <hint> :: <constraint>)

Examples:

PRED Hor(a, b) IS
    (E ax, bx, y :: a = (ax, y) AND b = (bx, y))
END;

FUNC y = Half(x) IS
    2 * y = x
END;
The DiGraph Interface

MODULE DiGraph;

PROC Node(c);

PROC Curved1(a,b,c,d);
Stroking with a Calligraphic Pen

MODULE PenStroke;
PROC Curve (a, b, c, d, p, q);

Spline Points

Control Points for pen width and orientation.
System Architecture

Compile-Time:

Cmd → Parser → AST → Compiler → Bytestream

Run-Time:

Bytestream → Run-Time → Display

Solver

Numeric Solver
Solving Constraints

Two Phases:
Symbolic solving (s)
Numeric solving (n)

Compile-time
(s) Unpack — convert constraint to simple normal form
(s) Preprocess — reduce number of unknowns

Run-time
(s) Separate structural (pair) and numeric constraints
(s) Solve pair constraints (unification closure)
(s) Repack — eliminate unknowns and constraints
(n) Solve numeric constraints (Newton's method)
Difficulties with Numeric Solving

Hints were lost during unpacking, preprocessing, repacking
  - Implement steps carefully so hints are preserved

Ordinary Newton willing to move large distances
  - Ensure each Newton step is as small as possible

Ordinary Newton unreliable on redundant systems
  - Modify Gaussian elimination to use only the "well-conditioned part" of matrix and ignore ill-conditioned part

Difficult to know when to terminate Newton iteration
  - Determine error threshold by estimating roundoff error
Conclusions

Juno-2 shows fast constraint-solving is possible with a constraint language that is:
- highly extensible \( \Rightarrow \) easy to define new constraints
- fully declarative \( \Rightarrow \) avoid imperative computations

Generality of interface is daunting for new users:
- PostScript drawing model vs. object drawing model
- Power of full imperative language is often overkill
- Using constraints still takes careful thinking

Using Juno-2 is fun!