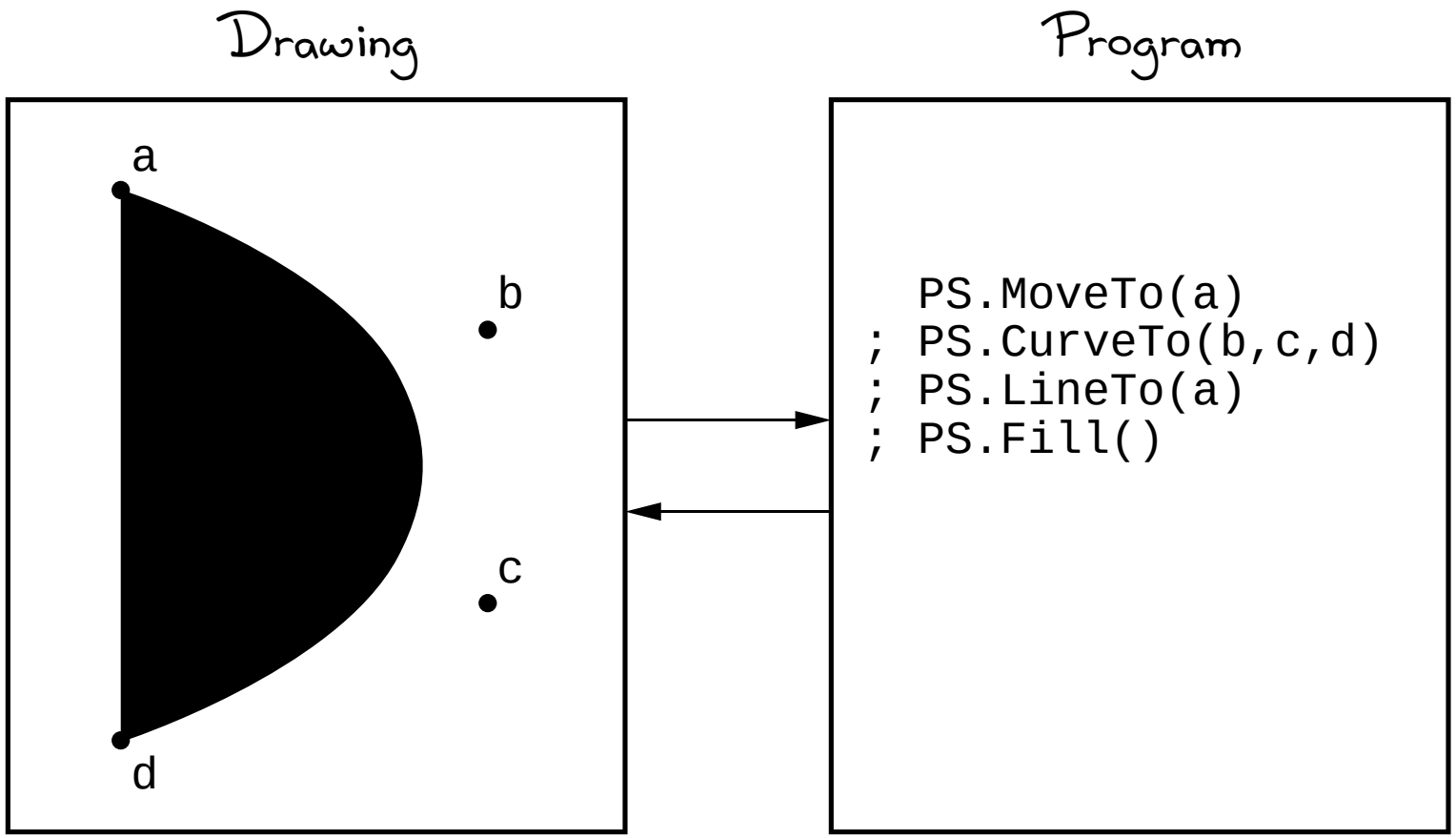
The Juno-2 Constraint-Based Drawing Editor

Allan Heydon
Greg Nelson
Digital Equipment Corporation
Systems Research Center (SRC)
http://www.research.digital.com/SRC/home.html

Issues in Constraint-Based Drawing

Issue	Juno-2 Approach
Representing constraints	Double-view editing
Specifying constraints	Entered directly with tools
Underconstrained systems	Hints
Redundant constraints	Ignore redundancy
Defining new constraints	Powerful extension language
Scale	Hierarchical structure

Double-View Editing



Constraint Solving

Syntax:

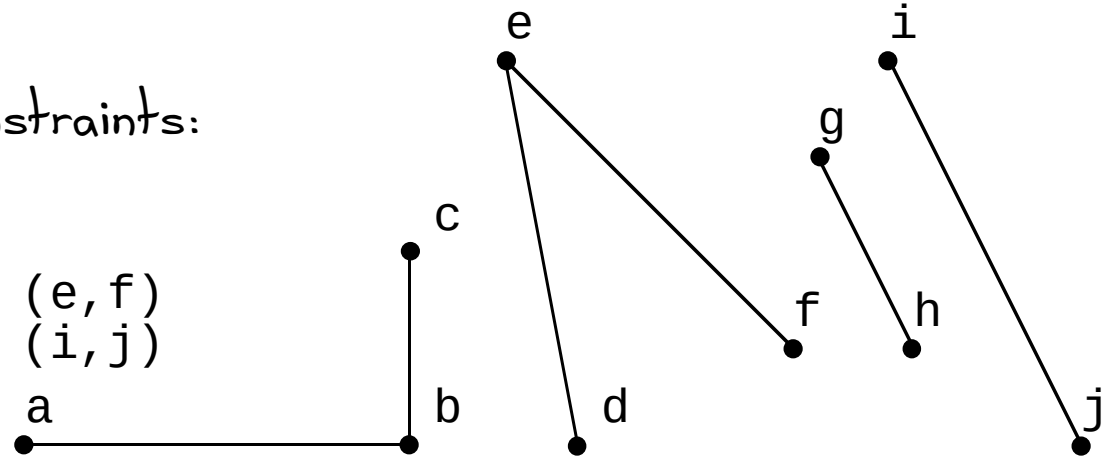
```
VAR <var> ~ <hint> IN
  <constraint> -> <statement>
END
```

Example:

```
VAR x ~ 1 IN
  x * x = 2 -> Print(x)
END
```

Built-In Constraints:

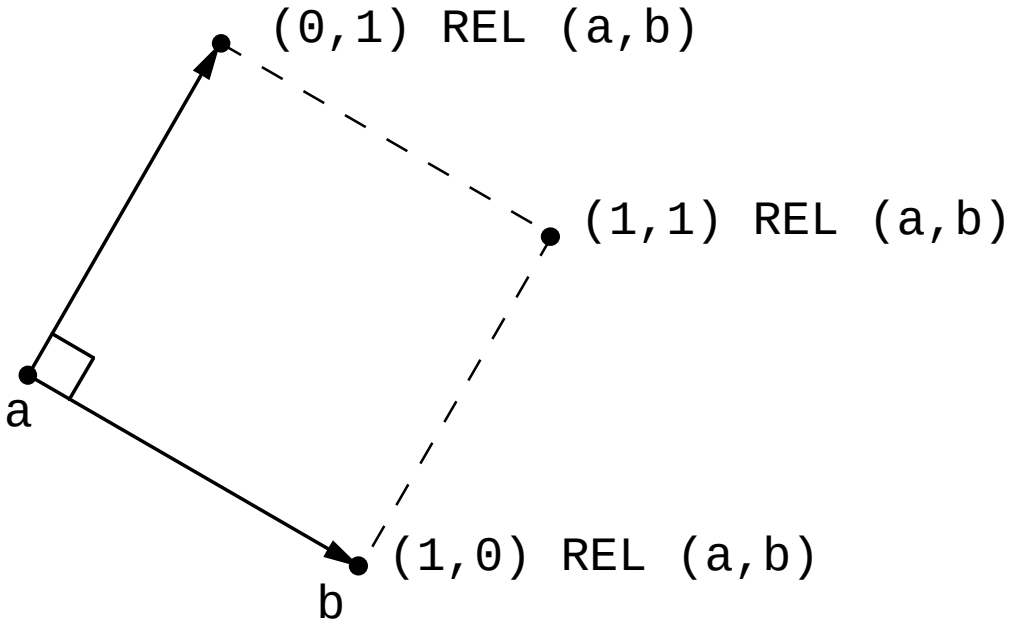
```
a HOR b
b VER c
(d,e) CONG (e,f)
(g,h) PARA (i,j)
```



The REL Function

`(x,y) 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:

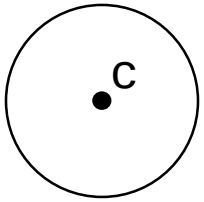
```
PRED Hor(a, b) IS
  (E ax, bx, y :: a = (ax, y) AND b = (bx, y))
END;

FUNC y = Half(x) IS
  2 * y = x
END;
```

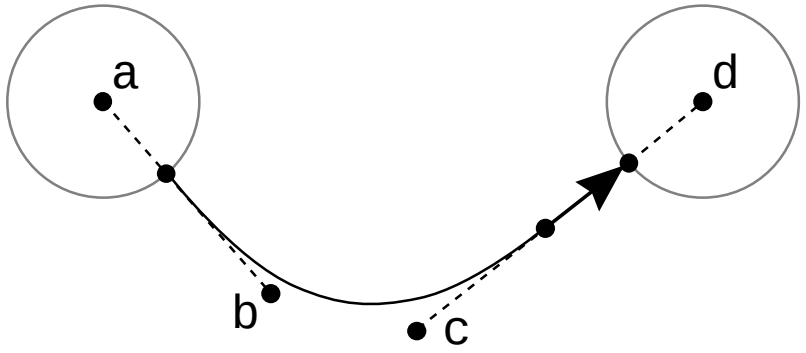
The DiGraph Interface

```
MODULE DiGraph;

PROC Node(c);
```




```
PROC Curved1(a,b,c,d);
```

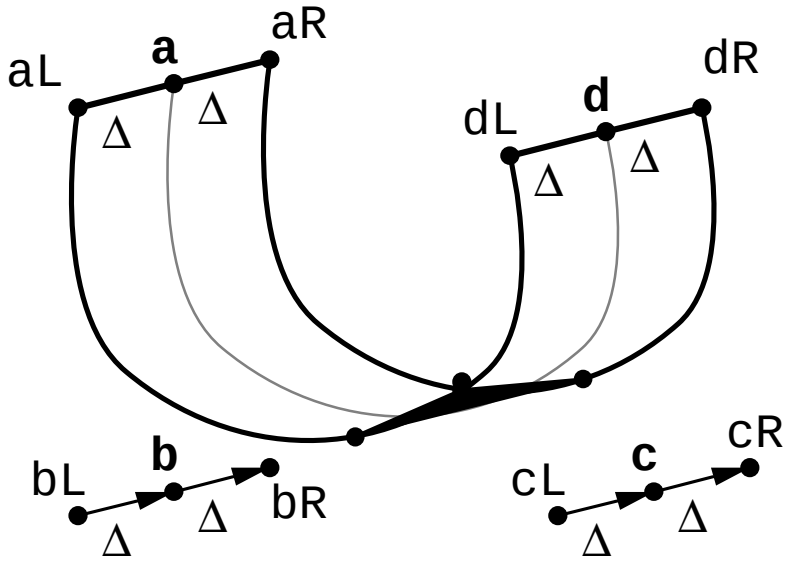


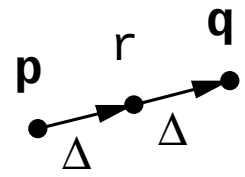
Stroking with a Calligraphic Pen

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

Spline Points

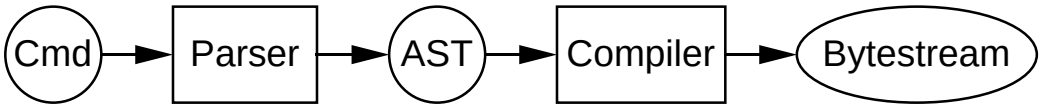
Control Points for pen width and orientation.



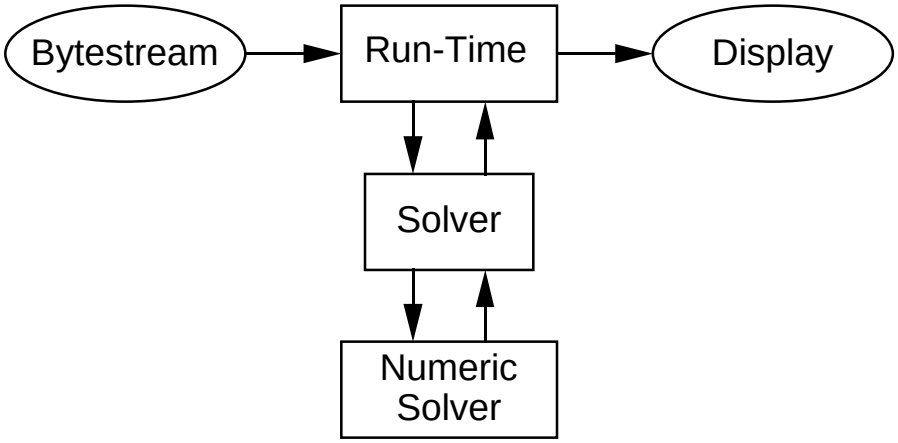


System Architecture

Compile-Time:



Run-Time:



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 Gassian 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 ⇒ easy to define new constraints
- fully declarative ⇒ 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!