# A normal form algorithm for piecewise functions

Jacques Carette

carette@mcmaster.ca

McMaster University

# Overview

- Observations
- Definitions
- Arithmetic and normal form
- Complexity
- Niceties
- Extensions

# Observations

- $\mathbb{R}$ is linearly ordered
  $\Rightarrow$ induces an order on domain decompositions

# Observations

- $\mathbb{R}$ is linearly ordered
  $\Rightarrow$ induces an order on domain decompositions
- $x$ is used *twice* in

$$f(x) = \begin{cases} -x & x < 0 \\ x & \text{otherwise.} \end{cases}$$

# Observations

- Algebraic properties of functions come (mostly) from those of the codomain.

$$+ : Y \times Y \to Y$$

induces a function

$$+ : (X \to Y) \times (X \to Y) \to (X \to Y)$$

by

$$(f + g)(x) = f(x) + g(x)$$

# Observations

- Algebraic properties of functions come (mostly) from those of the codomain.

  $$+ : Y \times Y \to Y$$

  induces a function

  $$+ : (X \to Y) \times (X \to Y) \to (X \to Y)$$

  by

  $$(f + g)(x) = f(x) + g(x)$$

- Composition **ruins** everything!

# Observations

Eager evaluation can be a problem:

$$f(x) = \begin{cases} 1/x & x < 0 \\ 23 & x = 0 \\ 2/x & \text{otherwise.} \end{cases}$$

# Definitions

**Definition.** *A* ***range partition*** $\mathcal{R}$ *of a linearly ordered set* $\Lambda$ *is a finite set* $B$ *of points* $\lambda_1 < \lambda_2 < \ldots < \lambda_n$*, along with the natural decomposition of* $\Lambda$ *into disjoint subsets* $\Lambda_1, \ldots, \Lambda_{n+1}$ *where*

$$\Lambda_1 := \{x \in \Lambda \mid x < \lambda_1\}$$

$$\Lambda_i := \{x \in \Lambda \mid \lambda_{i-1} < x < \lambda_i\}, i = 2, \ldots, n$$

$$\Lambda_{n+1} := \{x \in \Lambda \mid \lambda_n < x\}.$$

# Definitions

**Definition.** *A **piecewise expression** is a function from a range partition to a set $S$.*

**Example.** *Taking $\Lambda = \mathbb{R}$, the range partition $\mathcal{R}$ generated by $\{0\}$, and $S = \{x^2, x^3\}$ then $f : \mathcal{R} \to S$ defined by*

$$f(z) = \begin{cases} x^2 & z = \Lambda_1 \\ x^3 & z = 0 \\ x^3 & z = \Lambda_2, \end{cases}$$

*is a piecewise expression.*

# Definitions

**Definition.** *Let $S$ be a set of functions. Then a piecewise expression $f : \mathcal{R} \to S$ will be called a **piecewise operator**.*

Using $\tilde{S} = \{y \mapsto -y, y \mapsto 0, y \mapsto y\}$, we can write `abs` as the following piecewise operator:

$$a\tilde{b}s(x) = \begin{cases} y \mapsto -y & x < 0 \\ y \mapsto 0 & x = 0 \\ y \mapsto y & x > 0. \end{cases}$$

Of course we really want $f(x)(x)$.

# On notation

$$\begin{cases} h_1(x) & x < \lambda_1 \\ \beta_1 & x = \lambda_1 \\ h_2(x) & x < \lambda_2 \\ \beta_2 & x = \lambda_2 \\ \vdots & \vdots \\ \beta_n & x = \lambda_n \\ h_{n+1}(x) & \lambda_n < x \end{cases}$$

usual

$$\begin{cases} g_1 & x \in \Lambda_1 \\ g_2 & x = \lambda_1 \\ g_3 & x \in \Lambda_2 \\ g_4 & x = \lambda_2 \\ \vdots & \vdots \\ g_{2n} & x = \lambda_n \\ g_{2n+1} & x \in \Lambda_{n+1} \end{cases}$$

precise

# Definitions

**Definition.** *An **effective domain** $D$ is a pair $(F, \sim)$, where*

1. $F : O^n \to V$ *is a set of functions (of varied arity* $n$*)*
2. $\sim$ *is a function on* $F$ *that decides extensional equivalence.*

*Two functions* $f, g \in F$ *are said to be **extensionally equivalent** if* $\forall x \in O^n$*, either* $f$ *and* $g$ *are both defined and* $f(x) = g(x)$*, or neither* $f$ *nor* $g$ *are defined. Denoted* $f \simeq g$*.*

1. the functions in $F$ can be partial,
2. $\sim$ decides equivalence, not equality, and
3. $\sim$ is defined for $F$, not $O$ nor $V$.

# Arithmetic

See picture...

# Simplification

$$\begin{cases} y \mapsto -y & x < 0 \\ y \mapsto 0 & x = 0 \\ y \mapsto y & \text{otherwise.} \end{cases} + \begin{cases} y \mapsto y & x < 0 \\ y \mapsto 0 & x = 0 \\ y \mapsto -y & \text{otherwise.} \end{cases} =$$

$$\begin{cases} y \mapsto 0 & x < 0 \\ y \mapsto 0 & x = 0 \\ y \mapsto 0 & \text{otherwise.} \end{cases}$$

But also...

$$\begin{cases} y \mapsto 0 & x < 0 \\ y \mapsto y^2 & x = 0 \\ y \mapsto 0 & \text{otherwise.} \end{cases}$$

# Algorithm 1

- Prototype algorithm quickly in Maple
- Code it again with static types for correctness
-
```
type ('a,'b) endpiece = {fn : ('a -> 'b)} ;;
type ('a,'b) middlepiece =
    {left_fn:('a->'b); pt_fn : ('a -> 'b); right_pt : 'a}
and  ('a,'b) piecewise = (('a,'b) middlepiece) array *
                          ('a,'b) endpiece ;;
```

# Algorithm 1

```
let normalform (normal:('a->'b) -> ('a->'b))
    ((a,e):('a,'b) piecewise) : ('a,'b) piecewise =
    let pnormal y =
        {y with left_fn = normal y.left_fn; pt_fn = normal y.pt_fn}
    and canmerge a b = a.left_fn == a.pt_fn && a.pt_fn == b.left_fn
    and merge a b =
        {left_fn = a.left_fn; pt_fn = b.pt_fn; right_pt = b.right_pt}
    in
        let b = Array.map pnormal a
        and newe = {fn = normal (e.fn)}
        and j = ref 0
        and n = Array.length a
        in
```

# Algorithm 1

```
if n=0 then
     (b,newe)
else begin
     for i=1 to n-1 do
          if canmerge b.(!j) b.(i) then
               b.(!j) <- merge b.(!j) b.(i)
          else
               j := !j + 1;
     done;
     if b.(!j).left_fn==b.(!j).pt_fn &&
          b.(!j).pt_fn==newe.fn then
          (Array.sub b 0 !j, newe)
     else
          (Array.sub b 0 (!j+1), newe)
     end;;
```

# Normal form

- Theorem: preserves extensional equivalence

# Normal form

- Theorem: preserves extensional equivalence
- Is not a normal form

# Normal form

- Theorem: preserves extensional equivalence
- Is not a normal form
- Is a normal form when *functions* at the isolated points are related to one neighbour.

# Normal form

- Theorem: preserves extensional equivalence
- Is not a normal form
- Is a normal form when *functions* at the isolated points are related to one neighbour.
- In general: need to denest for a canonical form

# Normal form

- For a normal form: must evaluate
- ```
  let canmerge a b =
      ((a.left_fn = b.left_fn) &&
          (a.left_fn a.right_pt == a.pt_fn a.right_pt))
  ```
- Need $\sim$ on the codomain
  ```

# Complexity

- Previous work: based on `step` function
- Exponential complexity in number of breakpoints
- Ours: linear in number breakpoints
- But cost can still be dominated by base arithmetic

# Niceties

- Normalization of user input
- Left-to-right semantics

# Extensions

- General linearly ordered spaces
- Piecewise functions with mixed open, closed, clopen intervals
- Spaces given by finite decidable symbolic predicates
- Efficient denesting