A paradigm for interval-aware programming

Moritz Beutel, Robert Strzodka
Institute of Computer Engineering (ZITI), Heidelberg University, Germany

March 1, 2023 · Conference for Next Generation Arithmetic (CoNGA) 2023, Singapore

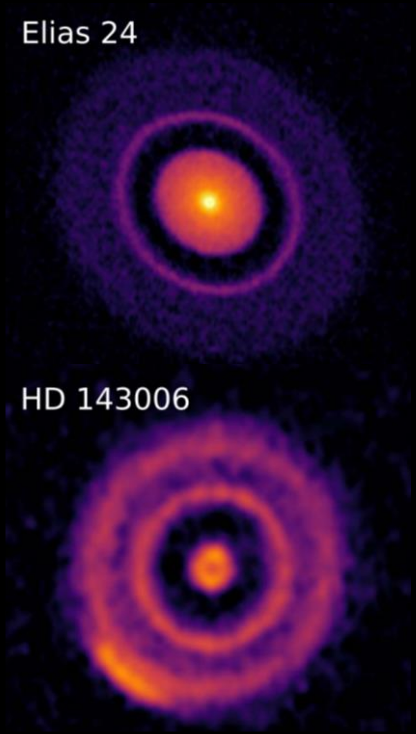
Motivation

Study protoplanetary growth with a stochastic representative particle method.

Problem:

n particles ➔ n^2 matrix of interaction rates:

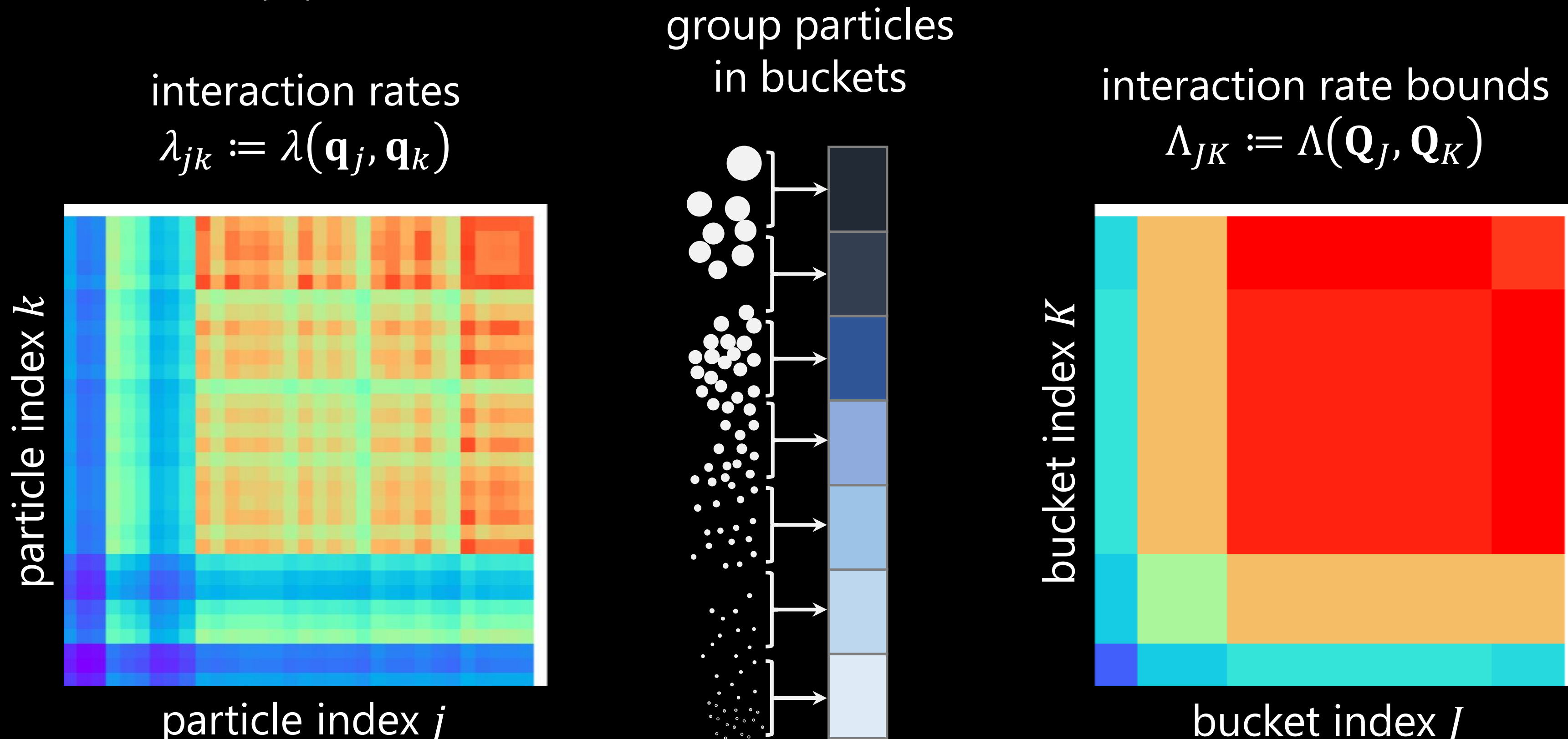
$$\begin{bmatrix} \lambda_{11} & \lambda_{12} & \lambda_{13} & \cdots & \lambda_{1n} \\ \lambda_{21} & \lambda_{22} & \lambda_{23} & \cdots & \lambda_{2n} \\ \lambda_{31} & \lambda_{32} & \lambda_{33} & \cdots & \lambda_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda_{n1} & \lambda_{n2} & \lambda_{n3} & \cdots & \lambda_{nn} \end{bmatrix}$$



dust rings observed by DSHARP survey, reproduced from Andrews et al., 2018 [1]

Bucketing

Beutel et al., in prep.



Set arithmetic

Arithmetic operations can be defined for set-valued arguments.

$$\sin\,[0,\pi) = [0,1]$$

$$f(\mathcal{S}) \coloneqq \{f(x) \mid x \in \mathcal{S}\}$$

('*set extension*')

Interval arithmetic

interval notation:
$X \equiv [X^-, X^+]$

Arithmetic operations can be defined for intervals.

Examples:

$$
\begin{aligned}
A + B &:= [A^- + B^-, A^+ + B^+] \\
-A &:= [-A^+, -A^-] \\
\sqrt{A} &:= \left[\sqrt{A^-}, \sqrt{A^+}\right]
\end{aligned}
$$

$$
\begin{aligned}
[5,7] + [0,1] &= [5,8] \\
-[0,1] &= [-1,0] \\
\sqrt{[4,9]} &= [2,3]
\end{aligned}
$$

('*interval extensions*')

‘Fundamental Theorem of Interval Arithmetic’

Let a function

$$f\colon \mathbb{R} \to \mathbb{R},\ x \mapsto f(x)$$

be composed of interval-extensible operations.

set of all intervals in $\mathbb{R}$

Construct a function

$$F\colon \overbrace{[\mathbb{R}]} \to [\mathbb{R}],\ X \mapsto F(X)$$

by replacing operations with their
interval extensions.

Then, F is an *interval extension* of f:

$$\forall X \in [\mathbb{R}]\ \forall x \in X\colon f(x) \in F(X).$$

say,

$$f(x) = x^2 - 2x$$
$$f(x) = (x-1)^2 - 1$$

algebraically equivalent

$$F(X) = X^2 - 2X$$
$$F(X) = (X-1)^2 - 1$$

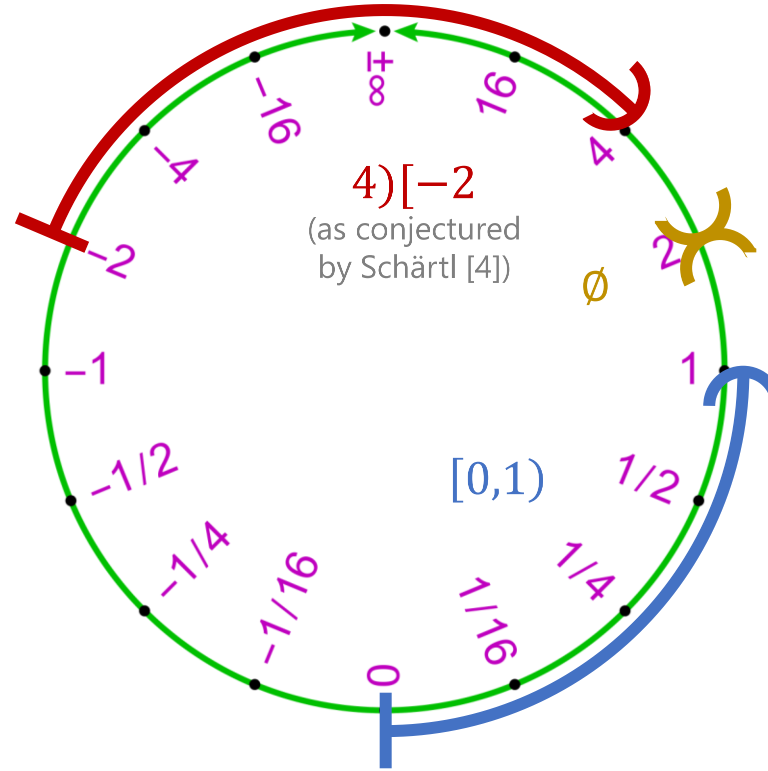
better results
(tighter intervals)

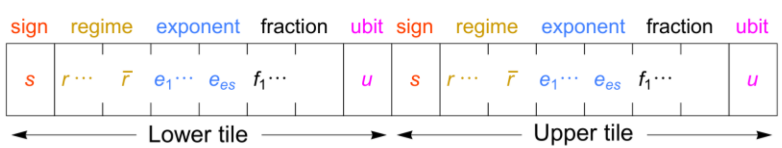
➔ dependency problem

Posits and valids

Posits were originally accompanied by *valids:*
Gustafson & Yonemoto, 2017 [2]

- posit arithmetic (and IEEE 754 floats): non-rigorous but fast
- valid arithmetic (and interval arithmetic): more expensive, but rigorous bounds





illustrations reproduced from Gustafson, 2017 [3], annotations mine

Goal:
'two modes of operation, selectable by the user'
Gustafson, 2017 [3]

Interval-aware code

$$f(x) = (x-1)^2 - 1$$

```
template <typename T>
T f(T x) {
  return square(x - 1) - 1;
}
```

```
auto x = 1.;
auto y = f(x);  // -1
```

```
auto X = Interval{ 0, 1 };  // [0,1]
auto Y = f(X);  // [-1,0]
```

```
auto p = Posit(1);
auto q = f(p);  // -1
```

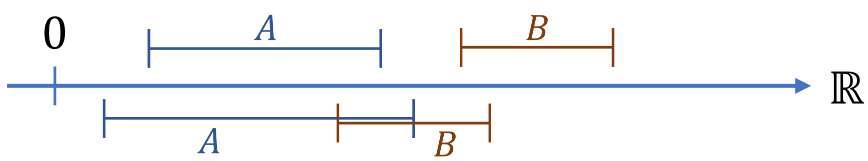
```
auto V = Valid{ 0,open, 0,open };  // ∅
auto W = f(V);  // ∅
```

Interval-aware code with branches

$$\max(a, b) \coloneqq \begin{cases} b & \text{if } a < b \\ a & \text{otherwise} \end{cases}$$

What could '$A < B$' mean for sets A, B?

- $\forall a \in A, b \in B\colon a < b$
- $\exists a \in A, b \in B\colon a < b$

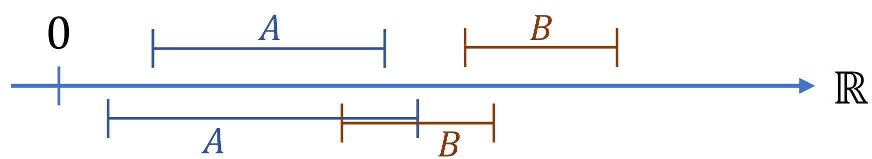


Interval-aware code with branches

$$\max(a, b) \coloneqq \begin{cases} b & \text{if } a < b \\ a & \text{otherwise} \end{cases}$$

What could '$A < B$' mean for sets A, B?

- '$\text{certainly } A < B$'
- '$\text{possibly } A < B$'



Implementing max

$$\max(a,b) \coloneqq \begin{cases} b & \text{if } a < b \\ a & \text{otherwise} \end{cases}$$

```
template <typename T>
T max(T a, T b) {
  T x;
  if (a < b) {
    x = b;
  }
  else {
    x = a;
  }
  return x;
}
```

A	B	fiducial result $\mathrm{Max}(A,B)$	'possibly' semantics $\texttt{max}(A,B)$	$\texttt{max}(B,A)$	'certainly' semantics $\texttt{max}(A,B)$	$\texttt{max}(B,A)$
$[0,2]$	$[3,6]$	$[3,6]$	$[3,6]$	$[3,6]$	$[3,6]$	$[3,6]$
$[2,4]$	$[3,6]$	$[3,6]$	$[3,6]$	$[2,4]$	$[2,4]$	$[3,6]$
$[4,5]$	$[3,6]$	$[4,6]$	$[3,6]$	$[4,5]$	$[4,5]$	$[3,6]$

too wide

incorrect

Relational identities

implicitly relied on by `else` clause

$$A \neq B \quad \Leftrightarrow \quad \neg(A = B)$$

$$A < B \quad \Leftrightarrow \quad \neg(A \geq B) \qquad \text{(complementarity)}$$

$$\mathrm{A} = B \quad \Leftrightarrow \quad \neg(A < B \lor A > B) \qquad \text{(totality)}$$

None of these are equivalent for set-valued arguments with either 'possibly' or 'certainly' semantics.

(check counterexample $A = B = [0,1]$)

Let's fix this.

Representing ambiguity

What *should* '$A < B$' mean for sets A, B?

Remember the *set extension*:

$$A < B = \underbrace{\{(a < b) \mid a \in A, b \in B\}}$$

one of $\emptyset$, {`false`}, {`true`}, or {`false`, `true`}

elements of the *powerset* of {`false`, `true`}

Boolean powerset logic

Two-element Boolean algebra $\mathbb{B} \coloneqq \{\texttt{false}, \texttt{true}\}$

Powerset of $\mathbb{B}$ $\mathcal{P}(\mathbb{B}) = \big\{\emptyset, \{\texttt{false}\}, \{\texttt{true}\}, \{\texttt{false}, \texttt{true}\}\big\}$

$\mathcal{P}(\mathbb{B})$ is a *four-valued logic*:

- logical connectives $\land$, $\lor$, $\neg$ given by set extension
- usual logical identities apply
  (associativity, commutativity, distributivity, De Morgan's laws)

Relational identities

$$A \neq B \quad \Leftrightarrow \quad \neg(A = B)$$

$$A < B \quad \Leftrightarrow \quad \neg(A \geq B)$$ (complementarity)

$$\mathrm{A} = B \quad \Leftrightarrow \quad \neg(A < B \vee A > B)$$ (totality)

All of these are equivalent for set-valued arguments with $\mathcal{P}(\mathbb{B})$-valued relational predicates.

Implementing max with set-valued logic

$$\max(a, b) \coloneqq \begin{cases} b & \text{if } a < b \\ a & \text{otherwise} \end{cases}$$

```cpp
template <typename T>
T max(T a, T b) {
  T x;

  if (a < b) {

    x = b;
  }
  else {

    x = a;
  }
  return x;
}
```

```cpp
template <typename T>
T max3(T a, T b) {
  auto x = T{ };
  auto c = (a < b);
  if (possibly(c)) {
    auto bc = constrain(b, c);
    assign_partial(x, bc);
  }
  if (possibly(!c)) {
    auto ac = constrain(a, !c);
    assign_partial(x, ac);
  }
  return x;
}
```

Implementing max with set-valued logic

$$\max(a,b) \coloneqq \begin{cases} b & \text{if } a < b \\ a & \text{otherwise} \end{cases}$$

```
template <typename T>
T max(T a, T b) {
  T x;

  if (a < b) {

    x = b;
  }
  else {

    x = a;
  }
  return x;
}
```

```
template <typename T>
T max3(T a, T b) {
  auto x = T{ };
  auto c = (a < b);
  if (possibly(c)) {
    auto bc = constrain(b, c);
    assign_partial(x, bc);
  }
  if (possibly(!c)) {
    auto ac = constrain(a, !c);
    assign_partial(x, ac);
  }
  return x;
}
```

value initialisation

Implementing max with set-valued logic

$$\max(a,b) \coloneqq \begin{cases} b & \text{if } a < b \\ a & \text{otherwise} \end{cases}$$

```cpp
template <typename T>
T max(T a, T b) {
  T x;

  if (a < b) {

    x = b;
  }
  else {

    x = a;
  }
  return x;
}
```

```cpp
template <typename T>
T max3(T a, T b) {
  auto x = T{ };
  auto c = (a < b);
  if (possibly(c)) {
    auto bc = constrain(b, c);
    assign_partial(x, bc);
  }
  if (possibly(!c)) {
    auto ac = constrain(a, !c);
    assign_partial(x, ac);
  }
  return x;
}
```

$\mathcal{P}(\mathbb{B})$ value

projection

possibly:

$\mathcal{P}(\mathbb{B}) \to \mathbb{B}$

$\mathcal{B} \mapsto (\texttt{true} \in \mathcal{B})$

$\mathcal{B}$	POSSIBLY($\mathcal{B}$)
$\emptyset$	false
{false}	false
{true}	true
{false, true}	true

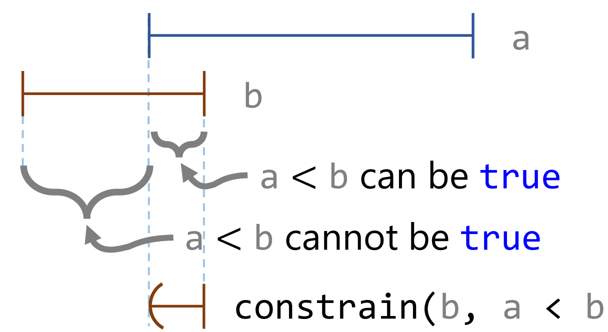
Implementing max with set-valued logic

$$\max(a, b) := \begin{cases} b & \text{if } a < b \\ a & \text{otherwise} \end{cases}$$

```cpp
template <typename T>
T max(T a, T b) {
  T x;

  if (a < b) {

    x = b;
  }
  else {

    x = a;
  }
  return x;
}
```

```cpp
template <typename T>
T max3(T a, T b) {
  auto x = T{ };
  auto c = (a < b);
  if (possibly(c)) {
    auto bc = constrain(b, c);
    assign_partial(x, bc);
  }
  if (possibly(!c)) {
    auto ac = constrain(a, !c);
    assign_partial(x, ac);
  }
  return x;
}
```

constrain argument with branch condition



Implementing max with set-valued logic

$$\max(a,b) \coloneqq \begin{cases} b & \text{if } a < b \\ a & \text{otherwise} \end{cases}$$

```cpp
template <typename T>
T max(T a, T b) {
  T x;

  if (a < b) {

    x = b;
  }
  else {

    x = a;
  }
  return x;
}
```

```cpp
template <typename T>
T max3(T a, T b) {
  auto x = T{ };
  auto c = (a < b);
  if (possibly(c)) {
    auto bc = constrain(b, c);
    assign_partial(x, bc);
  }
  if (possibly(!c)) {
    auto ac = constrain(a, !c);
    assign_partial(x, ac);
  }
  return x;
}
```

partial (additive) assignment

x = [3,5]

`assign_partial(x,z)`

z = [4,6)

x = [3,6)

Testing max3: empty set

```cpp
template <typename T>
T max3(T a, T b) {
  auto x = T{ };
  auto c = (a < b);
  if (possibly(c)) {
    auto bc = constrain(b, c);
    assign_partial(x, bc);
  }
  if (possibly(!c)) {
    auto ac = constrain(a, !c);
    assign_partial(x, ac);
  }
  return x;
}
```

false

false

$x = \emptyset$

return

$a = [4,5]$

$b = \emptyset$

$c = (a < b) = \emptyset$

Testing max3: overlapping intervals

```
template <typename T>
T max3(T a, T b) {
  auto x = T{ };
  auto c = (a < b);
  if (possibly(c)) {        ← true
    auto bc = constrain(b, c);
    assign_partial(x, bc);
  }
  if (possibly(!c)) {       ← true
    auto ac = constrain(a, !c);
    assign_partial(x, ac);
  }
  return x;
}
```

$x = \emptyset$

↓

$x = (4,6)$

↓

$x = [4,6)$

↓

return

$a = [4,5]$ $b = [3,6)$

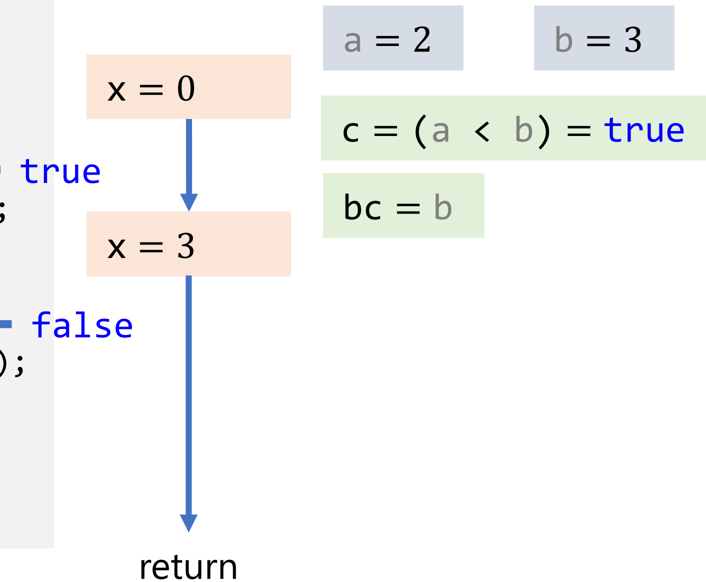
$c = (a < b) = \{\texttt{false}, \texttt{true}\}$

$bc = \{b' | b' \in b, \text{POSSIBLY } a < b'\} = (4,6)$

$ac = \{a' | a' \in a, \text{POSSIBLY } \neg(a' < b)\} = [4,5]$

Testing max3: numbers

```cpp
template <typename T>
T max3(T a, T b) {
  auto x = T{ };
  auto c = (a < b);
  if (possibly(c)) {
    auto bc = constrain(b, c);
    assign_partial(x, bc);
  }
  if (possibly(!c)) {
    auto ac = constrain(a, !c);
    assign_partial(x, ac);
  }
  return x;
}
```



Optimal results for max3

domain type	A	B	fiducial result $\mathrm{Max}(A,B)$	experimental results $\texttt{max3}(A,B)$	$\texttt{max3}(B,A)$
intervals	$[2,4]$	$[3,6]$	$[3,6]$	$[3,6]$	$[3,6]$
intervals	$[4,5]$	$[3,6]$	$[4,6]$	$[4,6]$	$[4,6]$
valids	$[2,4]$	$(3,\infty)$	$(3,\infty)$	$(3,\infty)$	$(3,\infty)$
valids	$[2,4]$	$[-\infty,6]$	$[2,6]$	$[2,6]$	$[2,6]$
valids	$[2,4]$	$3](6$	$[2,\infty)$	$[2,\infty)$	$[2,\infty)$

intervals library

https://github.com/mbeutel/intervals

- portable C++20
- traditional interval arithmetic (closed intervals) with `float`, `double`
- interval arithmetic with discrete types (integers, iterators)
- $\mathcal{P}(\mathbb{B})$ logic, Boolean projections, powerset arithmetic (`bool`, `enum`)

Experimentally adapted for posits and valids using the posit/valid implementation of A. Schärtl [4] which is part of the *aarith* library (Keszöcze et al., 2021 [5]).

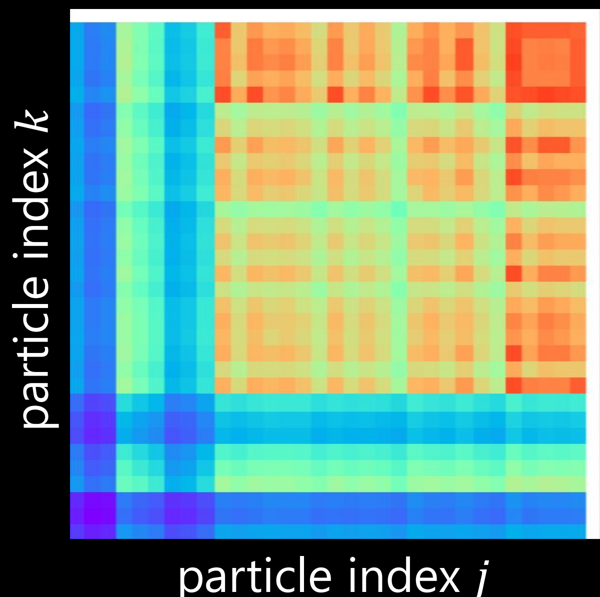
Bucketing

Beutel et al., in prep.

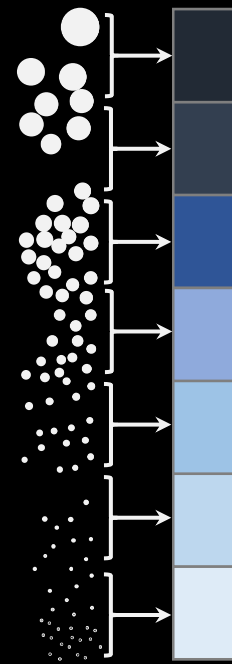
reuse interval-aware implementation of $\lambda(\mathbf{q}_j, \mathbf{q}_k)$

interaction rates

$\lambda_{jk} \coloneqq \lambda(\mathbf{q}_j, \mathbf{q}_k)$

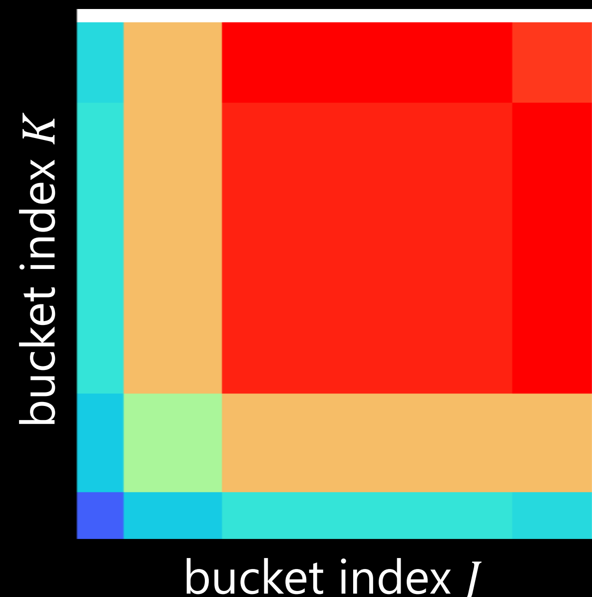


group particles in buckets



interaction rate bounds

$\Lambda_{JK} \coloneqq \Lambda(\mathbf{Q}_J, \mathbf{Q}_K)$



Summary

For set-valued operands, relational predicates should be set-valued.

Functions with branches can be made *interval-aware* by following the proposed programming pattern.

Try it out: https://github.com/mbeutel/intervals

References

[1] Andrews, S. M., Huang, J., Pérez, L. M., et al.: *The Disk Substructures at High Angular Resolution Project (DSHARP). I. Motivation, Sample, Calibration, and Overview.* ApJL, 869, L41 (2018)

[2] Gustafson, J.L., Yonemoto, I.: *Beating Floating Point at its Own Game: Posit Arithmetic.* Supercomputing Frontiers and Innovations 4(2), 16 (2017)

[3] Gustafson, J.L.: *Posit Arithmetic* (2017)

[4] Schärtl, A.: *Unums and Posits: A Replacement for IEEE 754 Floating Point?* M.Sc. thesis (2021)

[5] Keszöcze, O., Brand, M., Witterauf, M., Heidorn, C., Teich, J.: *Aarith: an arbitrary precision number library.* In: Proceedings of the 36th Annual ACM Symposium on Applied Computing. pp. 529–534. SAC ’21, Association for Computing Machinery, New York, NY, USA (2021)

this slide intentionally left blank