Some thoughts on designing a genomic query language

Limsoon Wong

Short talk for GeCo Workshop, Como, March 2019

Two perspectives on a query language

Surface syntax

- Easy to read, understand, & write queries correctly
- Sufficient power to express needed **queries**
- Prevent expensive queries

```
{ {x, z) | (x,y)∈A, (y',z)∈B, y = y' }

select (a.x, b.z)
from a in A, b in B
where a.y = b.y'
```

Abstract syntax

- Easy to analyze, manipulate, and optimize
- Easy to cater for extensions
- Sufficient power to express needed **algorithms**

```
U{ U{ if a.y = b.y'
          then  { (a.x, bz) } else {}
      | b ∈ B}
  |a ∈ A}
```

Compositionality & orthogonality are key principles for query language design

Compositionality & orthogonality in NRC

Types

$$s, t ::= \mid bool \mid b \mid s \times t \mid \{s\}$$

Expressions, *constructs are provided for each type orthogonally*

$$\frac{}{x^s : s} \qquad \frac{e_1 : s \quad e_2 : t}{(e_1, e_2) : s \times t} \qquad \frac{e : s \times t}{\pi_1\ e : s \quad \pi_2\ e : t}$$

$$\frac{}{true\ : bool} \qquad \frac{}{false\ : bool} \qquad \frac{e_1 : bool \quad e_2 : s \quad e_3 : s}{if\ e_1\ then\ e_2\ else\ e_3 : s}$$

$$\frac{}{\{\}^s : \{s\}} \qquad \frac{e : s}{\{e\} : \{s\}} \qquad \frac{e_1 : \{s\} \quad e_2 : \{s\}}{e_1 \bigcup e_2 : \{s\}}$$

$$\frac{e_1 : \{s\} \quad e_2 : \{t\}}{\cup\{e_1 \mid x^t \in e_2\} : \{s\}} \qquad \frac{e : \{s\}}{empty\ e : bool} \qquad \frac{e_1 : s \quad e_2 : s}{e_1 = e_2 : bool}$$

$\cup\{e_1 \mid x \in e_2\}$ means $f(o_1) \cup \ldots \cup f(o_n)$, where $f(x) = e_1$ and $\{o_1, \ldots, o_n\} = e_2$

Translating into comprehension syntax

$$\cup\{! e_1 \mid x \in e_2\} = \{! y \mid x \in e_2, y \in e_1\}$$

Translating from comprehension syntax

$$\{! e_1 \mid x \in e_2, \Delta\} = \cup\{! \{! e_1 \mid \Delta\} \mid x \in e_2\}$$
$$\{! e_1 \mid C, \Delta\} = \text{if } C \text{ then } \{! e_1 \mid \Delta\} \text{ else } \{\}$$
$$\{! e_1 \mid \} = \{! e_1\}$$

⇒Treat comprehension as a nice syntactic sugar

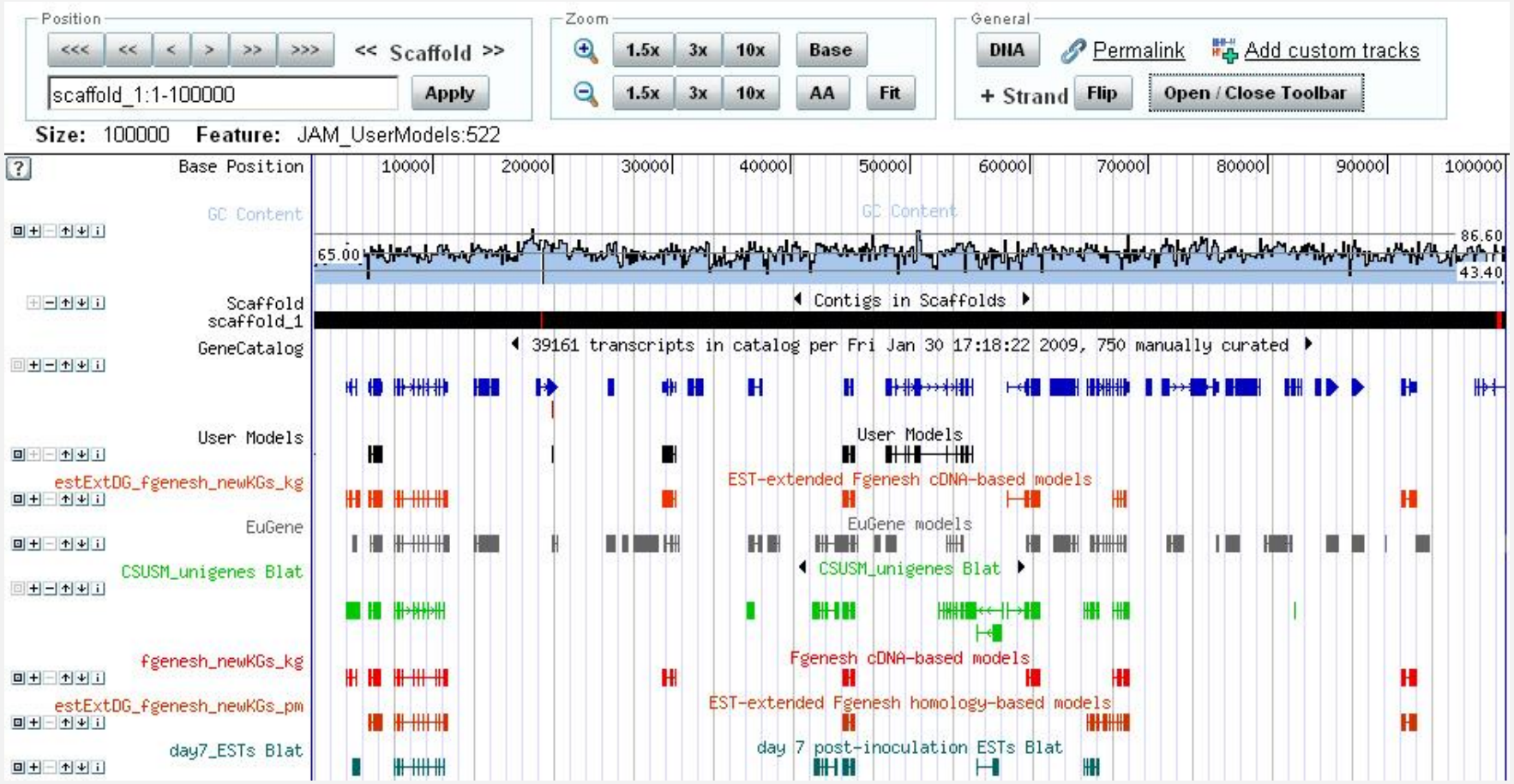
Genomic data

Loci

Tracks

Annotations

E.g. you see a row denoted “Base position”; this is the coordinates on a reference genome. The rest of the rows (e.g. “Gene catalog” and “day7_ESTs blat”) are “tracks” bearing different kinds of annotations. Each track corresponds to one kind of experiments, one kind of predictions, etc.; e.g. “day7_ESTS blat” are short stretches of RNAs from a day-7 sample that have been mapped (using a tool called “blat”) to specific positions on the reference genome.



Genomic data types

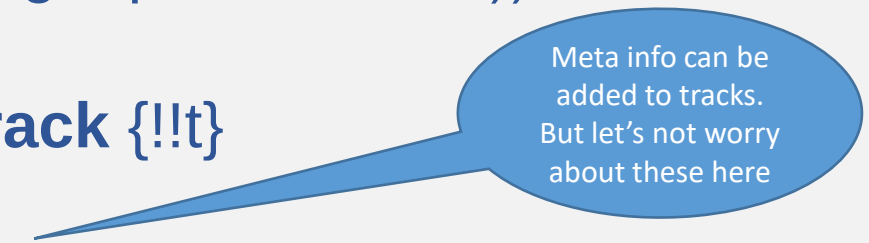
An **annotation** datatype !t and its subtype **landmark** !!t

of type t = (#name: string, #pval: real, …)

are represented as (#loc: Loc, #anno: (name: string, #pval: real, …))

A **track** datatype {!t} and its subtype **landmark track** {!!t}

are represented as { !t } and { !!t }

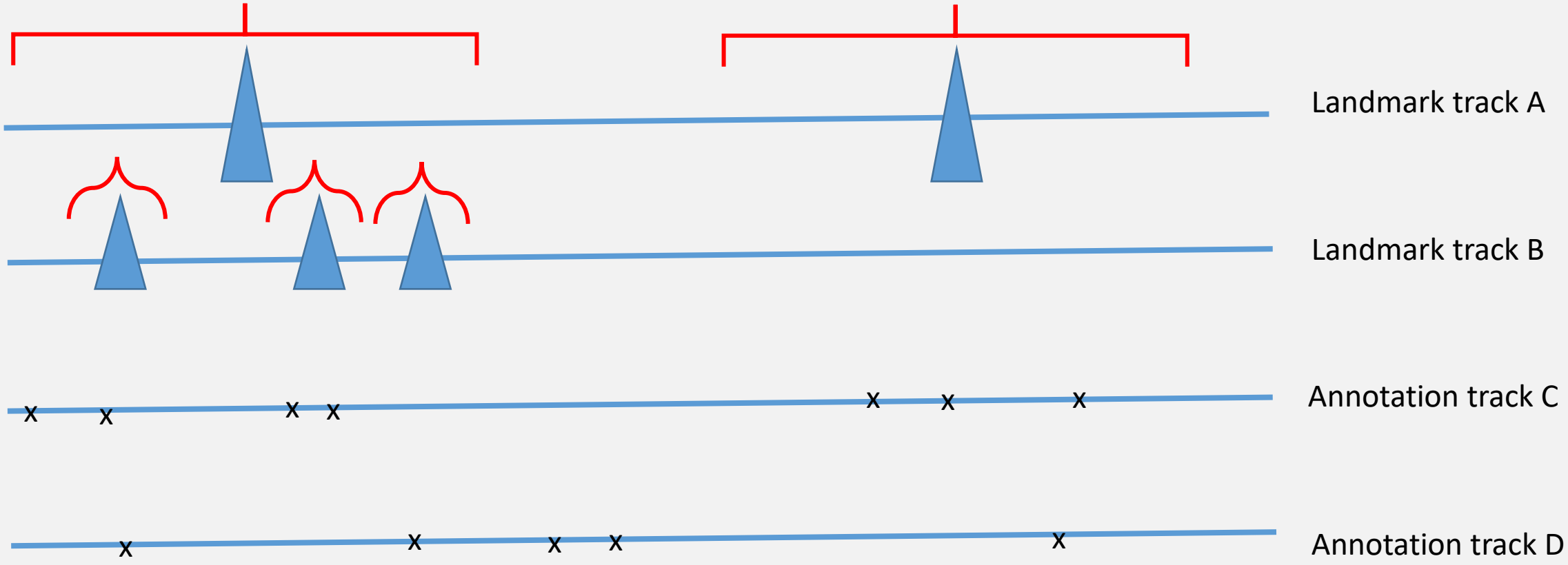


Equipped with some implicit / automatic normalizations / constraints, e.g. sorted by #loc, idempotency and non-overlapping loci on the same track

Landmarks on the same landmark track are non-overlapping, and all annotations can "see" no more than one landmark on the same landmark track

Landmarks can be used for organizing storage & distribution of annotations

Conceptual organization of annotation & landmark tracks



Some operations for the loci type

p before q
p overlap q
p near q …

} p, q : Loc

p can-see r

} p : Loc, r : !!t

satisfying “p can-see r whenever (p overlap or near q) & q = r.loc”,

plus maybe other convexity constraints to be thought up

Precise set of operations on loci (e.g. p is-nearer q_1 than q_2) is not important here

But a well-designed set of operations should constraint users from “bad” “expensive” queries, while providing sufficient expressive power for commonly needed queries

Operations on annotations and tracks

$$\frac{e_1 : \{!\ t_1\ \},\ e_2 : \{!\ t_2\}}{\cup\{!\ e_1 \mid x \in e_2\ \} : \{!\ t_1\ \}}$$

$$\frac{}{\{!\ \} : \{!\ t\ \}} \qquad \frac{e : !t}{\{!\ e\} : \{!\ t\}} \qquad \frac{e_1,\ e_2 : \{!\ t\ \}}{e_1 \cup e_2 : \{!\ t\ \}}$$

Semantics: Same as those for sets, except keep things sorted on #loc & maintains other constraints that we may come up with

Plus some set-track conversion ops & syntactic sugars

$$\frac{p : Loc,\ \ e : t}{!(\#loc: p, \#anno: e) : !t} \qquad \frac{e : !t}{e.loc : Loc,\ \ e.anno : t}$$

Ditto for !!t, but maybe ban !!(#loc, #anno)

Let's call this language NRC_{genome} in this talk

Common genomic queries in NRC_{genome}

"Extract from a track R, the annotations in a given region (e.g. 21q22.3) of the genome"

{! x | x ∈ R, x.loc overlap 21q22.3 }

"Extract from the HMMPFAM prediction track, those RBP predictions with pval < 1E-6"

{! x | x ∈ HMMPFAM, x.anno.name = "RBP", x.anno.pval < 1E-6 }

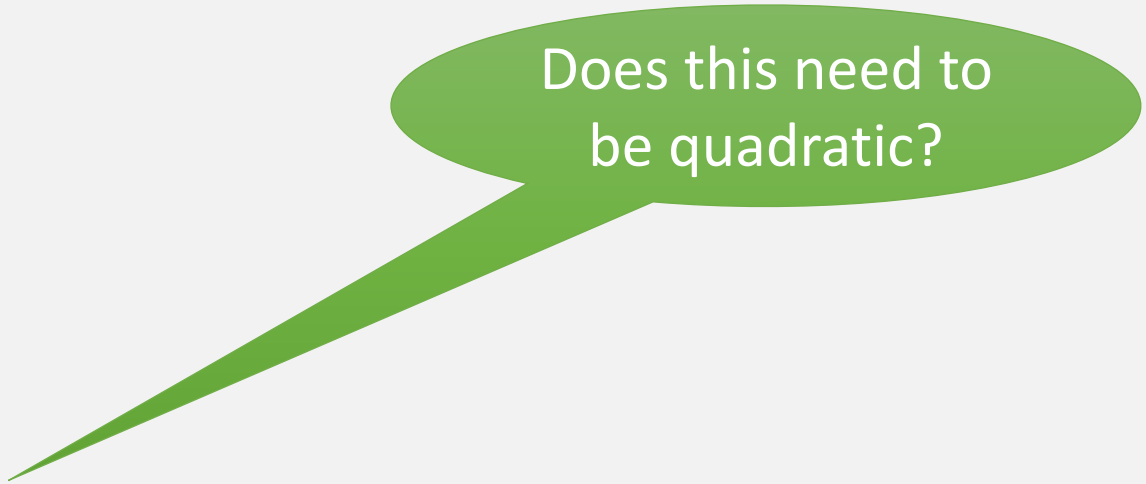
These queries operate on a single track

They can be executed efficiently, viz. O(n), in NRC_{genome}

Common genomic queries in NRC_{genome}

"Extract from the TP53 chip-seq track, those TP53 binding sites with pval < 1E-6 and are in promoters of genes"

{! x | y∈ GENES, x ∈ TP53,
x.loc before y.loc,
x.loc near y.loc,
x.anno.pval < 1E-6 }



This query operates on two tracks

Its "natural" complexity is O(|GENES| * |TP53|) in NRC_{genome}

Common genomic queries in NRC_{genome}

"Extract from the TP53 and the HDAC1 chip-seq tracks, those TP53 and HDAC1 binding sites that are closest to each other in the promoters of the same genes"

{! (#loc: g.loc, #anno: (#name: g.anno.name, #pval: 0, #tp53: u, #hdac1: v))
| g ∈ GENES,
  (u, v) ∈ closest { (x, y) | x ∈ TP53, x.loc near g.loc, x.loc before g.loc,
                             y ∈ HDAC1, y.loc near g.loc, y.loc before g.loc } }

This query has complexity O(|GENES| * |TP53| * |HDAC1|) in NRC_{genome}

Does this need to be cubic?

Common genomic queries in NRC_{genome}

"Extract from the TP53 and the HDAC1 chip-seq tracks, those TP53 and HDAC1 binding sites that are in the promoters of the same genes"

$$\{! u \mid u \in \{! (\#loc: g.loc, \#anno: (\#name: g.anno.name, \#pval: 0, \#tp53: \{ x \mid x \in TP53, x.loc \text{ near } g.loc, x.loc \text{ before } g.loc\}, \#hdac1: \{ y \mid y \in HDAC1, y.loc \text{ near } g.loc, y.loc \text{ before } g.loc\})) \mid g \in GENES \}, u.anno.tp53 \neq \{! \}, u.anno.hdac1 \neq \{! \}\}$$

This query has complexity $O(|GENES| * (|TP53| + |HDAC1|))$ in NRC_{genome}

Does this need to be quadratic?

What is needed? An idea

$$\frac{e : \{!\ t\},\ e_1 : \{!!\ t_1\},\ e_2 : \{!\ t_2\},\ \ldots,\ e_k : \{!\ t_k\}}{\cup\{!\ e \mid (x_1, X_2, \ldots, X_k) \in\in (e_1, e_2, \ldots, e_k)\} : \{!\ t\}}$$

Semantics

$$\cup\{!\ e \mid (x_1, X_2, \ldots, X_k) \in \mathbf{\{\ (x_1, \{\ x_2 \mid x_2 \in e_2,\ x_2.\text{loc can-see } x_1\},\ \ldots,\ \{\ x_k \mid x_k \in e_k,\ x_k.\text{loc can-see } x_1\})\ \mid x_1 \in e_1\ \}}\ \}$$

The part in bold is executed for each landmark, considering only annotations which can see that landmark (assuming these are stored with that landmark)

Common genomic queries revisited

"Extract from the TP53 chip-seq track those TP53 binding sites with pval < 1E-6 and are in promoters of genes"

{! x | y∈ GENES, x ∈ TP53,
x.loc before y.loc,
x.loc near y.loc,
x.anno.pval < 1E-6 }

GENES is a landmark track

{! x | (y, X) ∈∈ (GENES, TP53), x ∈ X,
x.loc before y.loc,
x.loc near y.loc,
x.anno.pval < 1E-6 }

Complexity is maybe O(|GENES| * 1% of |TP53|)

Common genomic queries revisited

"Extract from the TP53 and the HDAC1 chip-seq tracks, those TP53 and HDAC1 binding sites that are closest to each other in the promoters of the same genes"

```
{! (#loc: g.loc, #anno: (#name: g.anno.name, #pval: 0, #tp53: u, #hdac1: v))
| g ∈ GENES,
  (u, v) ∈ closest { (x, y) | x ∈ TP53, x.loc near g.loc, x.loc before g.loc,
                              y ∈ HDAC1, y.loc near g.loc, y.loc before g.loc } }
```

```
{! (#loc: g.loc, #anno: (#name: g.anno.name, #pval: 0, #tp53: u, #hdac1: v))
| (g, U, V) ∈∈ (GENES, TP53, HDAC1) ,
  (u, v) ∈ closest {(x,y) |  x ∈ U, x.loc near g.loc, x.loc before g.loc,
                             y ∈ V, y.loc near g.loc, y.loc before g.loc} }
```

Complexity is maybe O(|GENES| * (1% of |TP53| * 1% of |HDAC1|))

Common genomic queries revisited

"Extract from the TP53 and the HDAC1 chip-seq tracks, those TP53 and HDAC1 binding sites that are in the promoters of the same genes"

```
{! (#loc: g.loc, #anno: (#name: g.anno.name, #pval: 0,
        #tp53: { x | x ∈ TP53, x.loc near g.loc, x.loc before g.loc},
        #hdac1: { y | y ∈ HDAC1, y.loc near g.loc, y.loc before g.loc} ) )
| g ∈ GENES}
```

Is it necessary to process U and V twice?

```
{! (#loc: g.loc, #anno: (#name: g.anno.name, #pval: 0,
        #tp53: {! x |  x ∈ U, x.loc near g.loc, x.loc before g.loc},
        #hdac1: {! y | y ∈ V, y.loc near g.loc, y.loc before g.loc}))
| (g, U, V) ∈∈ (GENES, TP53, HDAC1) }
```

Complexity is maybe O(|GENES| * (1% of |TP53| + 1% of |HDAC1|)

A better idea?

$$e : \{!\ t\},\ e_1 : \{!!\ t_1\},\ e_2 : \{!\ t_2\},\ \ldots,\ e_k : \{!\ t_k\},\ \gamma_1 : \text{bool},\ \ldots,\ \gamma_k : \text{bool}$$

$$\cup\{!\ e \mid x_1 \in\in e_1 \text{ st } \gamma_1,\ X_2 \subseteq e_2\ \ni x_2 \text{ st } \gamma_2,\ \ldots,\ X_k \subseteq e_k\ \ni x_k \text{ st } \gamma_k\ \} : \{!\ t\ \}$$

$FV(\gamma_j) \setminus \{x_1, x_j\} \subseteq FV(\cup\{!\ e \mid x_1 \in\in e_1, X_2 \subseteq e_2\ \ni x_2 \text{ st } \gamma_2,\ \ldots,\ X_k \subseteq e_k\ \ni x_2 \text{ st } \gamma_k\ \}),$ and $FV(e) \cap \{x_2, \ldots, x_k\} = \{\ \}$

Semantics

$$\cup\{!\ e \mid (x_1, X_2, \ldots, X_k) \in \mathbf{\{\ (x_1, \{\ x_2 \mid x_2 \in e_2,\ x_2.\text{loc can-see } x_1,\ \gamma_2\},\ \ldots,\ \{\ x_k \mid x_k \in e_k,\ x_k.\text{loc can-see } x_1,\ \gamma_k\})\ \mid x_1 \in e_1,\ \gamma_1\ \}}\ \}$$

The part in bold is executed for each landmark, considering only annotations which can see that landmark (assuming these are stored with that landmark)

Common genomic queries revisited again

"Extract from the TP53 and the HDAC1 chip-seq tracks, those TP53 and HDAC1 binding sites that are in the promoters of the same genes"

```
{! (#loc: g.loc, #anno: (#name: g.anno.name, #pval: 0,
        #tp53: { x | x ∈ TP53, x.loc near g.loc, x.loc before g.loc},
        #hdac1: { y | y ∈ HDAC1, y.loc near g.loc, y.loc before g.loc} ) )
| g ∈ GENES}
```

```
{! (#loc: g.loc, #anno: (#name: g.anno.name, #pval: 0, #tp53: U, #hdac1: V))
| g ∈∈ GENES,
  U ⊆ TP53 ∋ u st u.loc near g.loc & u.loc before g.loc ,
  V ⊆ HDAC1 ∋ v st v.loc near g.loc, v.loc before g.loc }
```

Complexity is maybe O(|GENES| * (1% of |TP53| + 1% of |HDAC1|)

Common genomic queries revisited again

"Extract from the TP53 and the HDAC1 chip-seq tracks, those TP53 and HDAC1 binding sites that are closest to each other in the promoters of the same genes"

```
{! (#loc: g.loc, #anno: (#name: g.anno.name, #pval: 0, #tp53: u, #hdac1: v))
| g ∈ GENES,
  (u, v) ∈ closest { (x, y) | x ∈ TP53, x.loc near g.loc, x.loc before g.loc,
                              y ∈ HDAC1, y.loc near g.loc, y.loc before g.loc } }
```

```
{! (#loc: g.loc, #anno: (#name: g.anno.name, #pval: 0, #tp53: x, #hdac1: y))
| g ∈∈ GENES, U ⊆ TP53ɜ u st u.loc near g.loc & u.loc before g.loc,
  V ⊆ HDAC1 ɜ v st v.loc near g.loc & v.loc before g.loc ,
  (x, y) ∈ closest {(u,v) |  u ∈ U, v ∈ V} }
```

Complexity is maybe O(|GENES| * (1% of |TP53| * 1% of |HDAC1|))

And this idea? It is really a syntactic sugar

$$\frac{e : \{!\ t\ \},\ e_1 : \{!!\ t_1\ \},\ e_2 : \{!\ t_2\ \},\ \ldots,\ e_k : \{!\ t_k\ \},\ \gamma_1 : \text{bool},\ \ldots,\ \gamma_k : \text{bool}}{\cup\{!\ e \mid x_1 \in\in e_1\ \text{st}\ \gamma_1,\ x_2 \in e_2\ \text{st}\ \gamma_2,\ \ldots,\ x_k \in e_k\ \text{st}\ \gamma_k\ \} : \{!\ t\ \}}$$

$FV(\gamma_j) \setminus \{x_1, x_j\} \subseteq FV(\cup\{!\ e \mid x_1 \in\in e_1, x_2 \in e_2\ \text{st}\ \gamma_2,\ \ldots,\ x_k \in e_k\ \text{st}\ \gamma_k\ \})$

Semantics

$$\cup\{!\ e \mid (x_1, X_2, \ldots, X_k) \in \mathbf{\{\ (x_1, \{\ x_2 \mid x_2 \in e_2, x_2.\text{loc can-see } x_1, \gamma_2\}, \ldots, \{\ x_k \mid x_k \in e_k, x_k.\text{loc can-see } x_1, \gamma_k\})\ \mid x_1 \in e_1\ , \gamma_1\ \}},\ x_2 \in X_2, \ldots, x_k \in X_k\ \}$$

The part in bold is executed for each landmark, considering only annotations which can see that landmark (assuming these are stored with that landmark)

Common genomic queries revisited again

"Extract from the TP53 chip-seq track those TP53 binding sites with pval < 1E-6 and are in promoters of genes"

{! x | y∈ GENES, x ∈ TP53,
x.loc before y.loc,
x.loc near y.loc,
x.anno.pval < 1E-6 }

GENES is a landmark track

{! x | y ∈∈ GENES,
x ∈ TP53 st x.loc before y.loc &
x.loc near y.loc & x.anno.pval < 1E-6 }

Complexity is maybe O(|GENES| * 1% of |TP53|)

Implementing "synchronized" processing of multiple lists / tracks

lzip: $(t_1 \to \text{bool}) * (t_1 * t_2 \to \text{bool}) * (t_1 * t_2 \to \text{bool}) * (t_2 * t' \to t') * (t_1 * t' \to t') * (t' \to \{t\}) * t' * t'$
$\to \{t_1\} * \{t_2\} \to \{t\}$

```
lzip (sx, sy, ay, h, g, f, a, e)  ({}, Y) = f a
lzip (sx, sy, ay, h, g, f, a, e)  (X, {}) = f a
lzip (sx, sy, ay, h, g, f, a, e)  (x::X, y::Y) =
        if sx(x)
        then if sy(x, y)
             then if ay(x,y)
                  then lzip (sx, sy, ay, h, g, f, h(y, g(x, a)), e) (x::X, Y)
                  else lzip (sx, sy, ay, h, g, f, g(x, a), e) (x::X, Y)
             else f (g(x, a)) ∪ lzip (sx, sy, ay, h, g, f, e, e) (X, y::Y)
        else f a ∪ lzip (sx, sy, ay, h, g, f, e, e) (X, y::Y)
```

At every step, either x or y gets shifted. So complexity is $O(|X| + |Y| * \alpha)$, where α is complexdity of sx, sy, etc.

Implementing $\cup\{!\ e \mid x_1 \in\in e_1$ st γ_1, $X_2 \subseteq e_2\ \ni x_2$ st γ_2, …, $X_k \subseteq e_k\ \ni x_k$ st $\gamma_k\ \}$

$\cup\{!\ e \mid x_1 \in\in e_1$ st γ_1, $X_2 \subseteq e_2\ \ni x_2$ st $\gamma_2\ \} :=$

lzip(sx, sy, ay, h, g, f, ({!! },{! }), ({!! },{! })) (e_1, e_2) where

$sx(x_1) := \gamma_1$,

$ay(x_1, x_2)$:= x_2.loc can-see x_1 & γ_2,

$sy(x_1, x_2)$:= x_2.loc before x_1.loc or $ay(x_1, x_2)$,

$h(x_2, (X_1,X_2)) := (X_1, X_2 \cup \{!\ x_2\ \})$,

$g(x_1, (X_1,X_2)) := (X_1 \cup \{!!\ x_1\ \}, X_2)$,

$f(X_1,X_2) := \cup\{!\ e \mid x_1 \in X_1\}$;

Synchronized scan

lzip: $(t_1 \to bool) * (t_1 * t_2 \to bool) * (t_1 * t_2 \to bool) * (t_2 * t' \to t') * (t_1 * t' \to t') * (t' \to \{t\}) * t' * t'$
$\to \{t_1\} * \{t_2\} \to \{t\}$

```
lzip (sx, sy, ay, h, g, f, a, e)  ({}, Y) = f a
lzip (sx, sy, ay, h, g, f, a, e)  (X, {}) = f a
lzip (sx, sy, ay, h, g, f, a, e)  (x::X, y::Y) =
        if sx(x)
        then if sy(x, y)
                then if ay(x,y)
                        then lzip (sx, sy, ay, h, g, f, h(y, g(x, a)), e) (x::X, Y)
                        else lzip (sx, sy, ay, h, g, f, g(x, a), e) (x::X, Y)
                else f (g(x, a)) ∪ lzip (sx, sy, ay, h, g, f, e, e) (X, y::Y)
        else f a ∪ lzip (sx, sy, ay, h, g, f, e, e) (X, y::Y)
```

A nice property of $\cup\{!\ e \mid x_1 \in\in e_1$ st γ_1, $X_2 \subseteq e_2 \ni x_2$ st γ_2, …, $X_k \subseteq e_k \ni x_k$ st $\gamma_k\}$

$\cup\{!\ e \mid x_1 \in\in e_1$ st γ_1, $X_2 \subseteq e_2 \ni x_2$ st $\gamma_2\}$:=
lzip (…) (e_1, e_2) where …

```
lzip: (t1 → bool) * (t1 * t2 → bool) * (t1 * t2 → bool) * (t2 * t' → t') * (t1 * t' → t') * (t' → {t}) * t' * t'
                                  → {t1} * {t2} → {t}

lzip (sx, sy, ay, h, g, f, a, e)  ({}, Y) = f a
lzip (sx, sy, ay, h, g, f, a, e)  (X, {}) = f a
lzip (sx, sy, ay, h, g, f, a, e)  (x::X, y::Y) =
        if sx(x)
        then if sy(x, y)
             then if ay(x,y)
                  then lzip (sx, sy, ay, h, g, f, h(y, g(x, a)), e) (x::X, Y)
                  else lzip (sx, sy, ay, h, g, f, g(x, a), e) (x::X, Y)
             else f (g(x, a)) ∪ lzip (sx, sy, ay, h, g, f, e, e) (X, y::Y)
        else f a ∪ lzip (sx, sy, ay, h, g, f, e, e) (X, y::Y)
```

is a homomorphism on e_1. Thus

$\cup\{!\ e \mid x_1 \in\in \{!!\ o_1, \ldots, o_k\}$ st γ_1, $X_2 \subseteq e_2 \ni x_2$ st $\gamma_2\}$
= $\cup\{!\ e \mid x_1 \in\in \{!!\ o_1\}$ st γ_1, $X_2 \subseteq e_2 \ni x_2$ st $\gamma_2\} \cup \ldots \cup$
$\cup\{!\ e \mid x_1 \in\in \{!!\ o_k\}$ st γ_1, $X_2 \subseteq e_2 \ni x_2$ st $\gamma_2\}$

When annotations on track e2 are “clustered” (i.e. stored with) the specific landmarks on track e1 these annotations “can see”, each $\cup\{!\ e \mid x_1 \in\in \{!!\ o_j\}$ st γ_1, $X_2 \subseteq e_2 \ni x_2$ st $\gamma_2\}$ can be run in parallel on each cluster

Some optimization rules

$$\cup\{!\ \text{if } \varphi \text{ then e else } \{!\ \} \mid x_1 \in\in e_1 \text{ st } \gamma_1,\ x_2 \in e_2 \text{ st } \gamma_2,\ \ldots,\ x_k \in e_k \text{ st } \gamma_k\ \}$$

$$\Downarrow$$

$$\Downarrow\ x_j \in FV(\varphi) \text{ and } FV(\varphi) \cap \{x_1, \ldots, x_k\} \subseteq \{x_1, x_j\}$$

$$\Downarrow$$

$$\cup\{!\ e \mid x_1 \in\in e_1 \text{ st } \gamma_1,\ x_2 \in e_2 \text{ st } \gamma_2,\ \ldots,\ \mathbf{x_j \in e_j\ st\ \gamma_j\ \&}\ \varphi, \ldots,\ x_k \in e_k \text{ st } \gamma_k\ \}$$

And φ is a "positive" condition on loci in both rules

$$\cup\{!\ \cup\{!\ \text{if } \varphi \text{ then e else } \{!\ \}\ \mid x \in X_j\}$$

$$\mid x_1 \in\in e_1 \text{ st } \gamma_1,\ X_2 \subseteq e_2\ \ni x_2 \text{ st } \gamma_2,\ \ldots,\ X_k \subseteq e_k\ \ni x_2 \text{ st } \gamma_k\ \}$$

$$\Downarrow$$

$$\Downarrow\ x \in FV(\varphi) \text{ and } FV(\varphi) \cap \{x, x_1, X_2, \ldots, X_k\} \subseteq \{x_1, x\}$$

$$\Downarrow$$

$$\cup\{!\ \cup\{!\ e\ \mid x \in X_j\}$$

$$\mid x_1 \in\in e_1 \text{ st } \gamma_1,$$

$$X_2 \subseteq e_2\ \ni x_2 \text{ st } \gamma_2,\ \ldots,\ X_j \subseteq e_j\ \ni x_j \text{ st } \gamma_j\ \&\ \varphi[x_j/x], \ldots,\ X_k \subseteq e_k\ \ni x_2 \text{ st } \gamma_k\ \}$$

And …

$\cup\{! \cup\{! \text{ if } \varphi \text{ then e else } \{! \} \mid x_2 \in e_2\} \mid x_1 \in e_1\}$

$\Downarrow$

$\Downarrow e_1: !!t_1 \;\&\; e_2: !t_2 \;\&$

$\Downarrow x_1 \notin FV(e_2) \;\&\; FV(\varphi) \cap \{x_1,x_2\} = \{x_1,x_2\} \;\&$

$\Downarrow \varphi$ is a positive condition on loci of x_1,x_2

$\Downarrow$

$\cup\{! \cup\{! \text{ if } \varphi \text{ then e else } \{! \} \mid x_1 \in\in e_1 \text{ st true}, x_2 \in e_2 \text{ st true } \}$

So a user does not need to worry about when to use $\cup\{! \text{ e} \mid x_1 \in\in e_1, \ldots\}$

In fact, …

It is not necessary for a user to use {!, {!!, etc.

These can be inferred by a simple type system

And transformed into synchronized/parallel scans by an optimizer