universiteit hasselt

KNOWLEDGE IN ACTION

Relational Database Representation and Manipulation in DNA

Jan Van den Bussche

joint work with
Joris Gillis, Robert Brijder

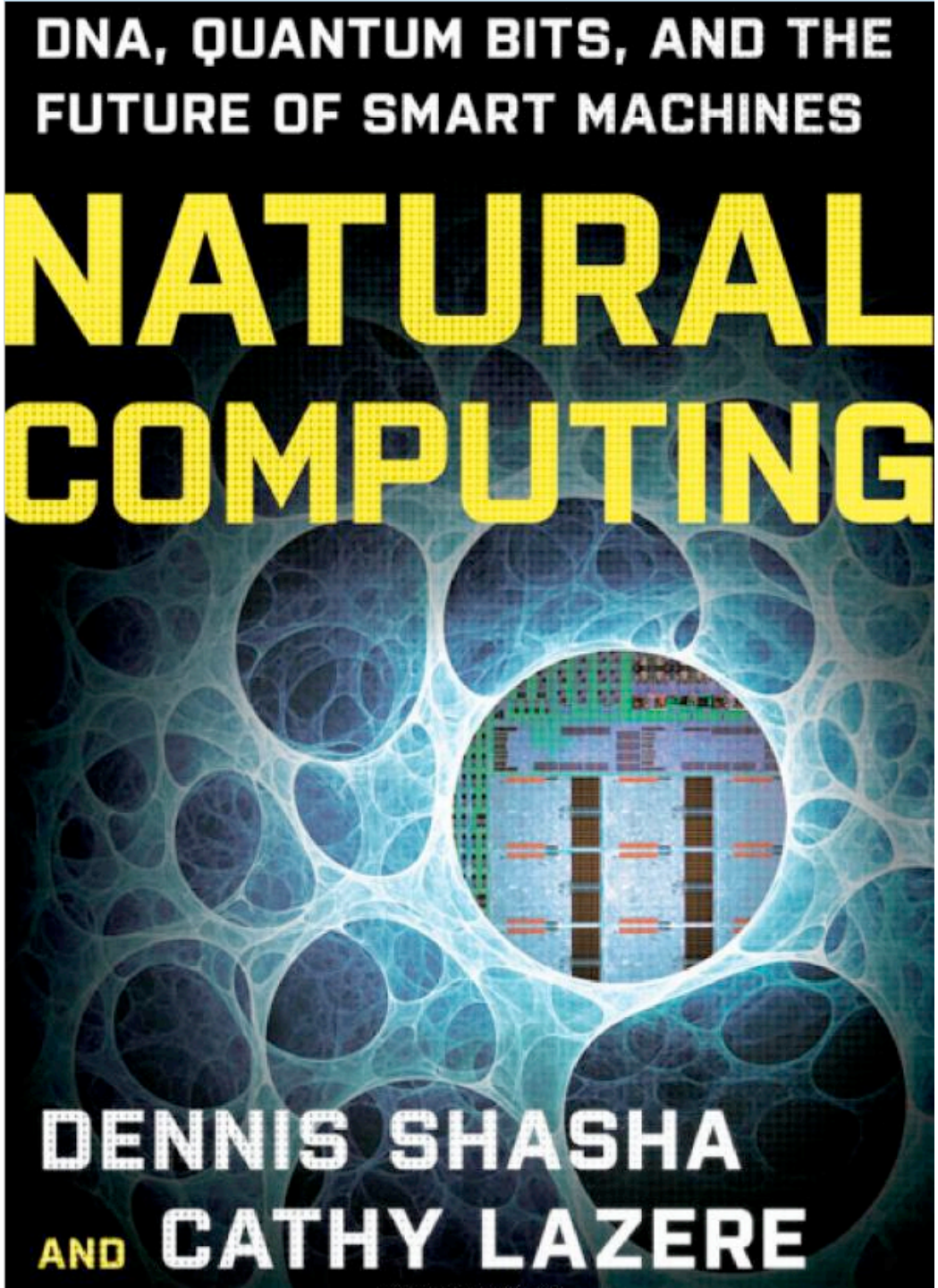
Hasselt University, Belgium

Natural Computing

1. Conventional computing, inspired by nature
   - Evolutionary systems, algorithms, programs
   - Parallel systems, swarm computing
2. Physics as a computation model
   - Analog computers
   - Quantum computing
3. "Wet" computing: use hardware from nature
   - ☞ DNA computing
   - Reprogrammed bacteria & viruses

DNA, QUANTUM BITS, AND THE FUTURE OF SMART MACHINES

NATURAL COMPUTING



DENNIS SHASHA
AND CATHY LAZERE

DNA Computing: What it is NOT

- Solving NP-complete problems
  - First DNA computing experiment solved a small instance of the Hamiltonian Path problem
  - [Adleman, Science 1994]
- Genetic engineering
  - DNA computing works with dead material
  - Synthetic DNA
- Bioinformatics
  - Conventional databases, algorithms to store, analyse genetic information

DNA Computing: What it IS

- Use synthetic DNA molecules as data carrier
- Programmed nanotechnology
- Computation on the DNA carried out by:
  - Biotechnology laboratory protocols
  - Enzymes
  - DNA itself: self-assembly
- Computation goes on in:
  - *In vitro:* Test tube (watery solution)
  - DNA chips, diamond surfaces
  - *In vivo* (smart medicine)

DNA Computing a vibrant field

- Two annual international meetings
  - International Conference on DNA Computing and Molecular Programming
  - Conference on Foundations of Nanoscience
- Diverse community
  - Experimental chemists, physicists
  - Theoretical computer scientists
  - Computer simulations
- Papers in Nature, Science
- Large gap between theory and practice

Use synthetic DNA molecules as data carrier

- In digital computers, all data is in strings of bits
  - 0 and 1
- Single-stranded DNA molecule:
  = string over the 4-letter alphabet {A,C,G,T}

Data storage in DNA

- Enormous capacity
  - Theoretical capacity ~ 455 EB per gram
  - ~ 2.2 PB per gram with reliable encode & decode
  - [Goldman et al., Nature 2013]
- Very robust
- Long term
  - 1000nds of years
  - Can be copied
- Archiving

Databases in DNA?

- We need much more than mere archival write/read
- Structured data
- Efficient and flexible access
- Logical data model
- Query language

☞ DNA computing

Talk Outline

1. Primer on databases
2. Representing tuples, relations in DNA
3. Doing relational algebra by DNA computing
4. DNAQL, the language
5. DNA complexes: the DNAQL data model
6. Typechecking
7. Expressive power of DNAQL

Relational database

- A collection of tables called relations
- Column headings are called attributes
- Rows are called tuples

Database schema

- Consists of:
  - Names of the relations
  - Attributes of each relation
- Example:
  - Likes(drinker,beer)
  - Visits(drinker,bar)
  - Serves(bar,beer)

Database instance

- Actual content (tuples) of the relations

Likes	
drinker	**beer**
John	Hoegaerden
John	Westmalle
Mary	Hoegaerden
Mary	Chouffe

Visits	
drinker	**bar**
John	Zur Laube
John	Albrecht
Mary	Albrecht

Serves	
bar	**beer**
Zur Laube	Duvel
Zur Laube	Westmalle
Zur Laube	Hoegaerden
Albrecht	Hoegaerden
Albrecht	Chouffe
Kugel	Westmalle

Relational algebra

- Operations applied to relations
- Compute new relations from given ones
- Answer queries to the database

  - selection (**σ**)
  - projection (**π**)
  - renaming (**ρ**)
  - union (∪)
  - set difference (**–**)
  - join (⋈)

Selection (σ)

- $\sigma_{drinker='John'}$: select tuples with specified value for given attribute

Likes	
drinker	**beer**
John	Hoegaerden
John	Westmalle
Mary	Hoegaerden
Mary	Chouffe

$\sigma_{drinker='John'}$ (Likes)	
drinker	**beer**
John	Hoegaerden
John	Westmalle

Projection (π)

- π_{beer} : select specified attribute(s)

$\sigma_{drinker='John'}$ (Likes)	
drinker	**beer**
John	Hoegaerden
John	Westmalle

$\pi_{beer}(\sigma_{drinker='John'}$ (Likes))
beer
Hoegaerden
Westmalle

Set difference (−)

- $R_1 - R_2$: all tuples from R_1 that are **not** in R_2
- "List all beers served by Albrecht that John does not like"

$\pi_{beer}(\sigma_{bar='Albrecht'}(Serves)) - \pi_{beer}(\sigma_{drinker='John'}(Likes))$

Join (⋈)

- Pairs up compatible tuples from two relations
- "List bars that serve a beer that John likes"

$$\sigma_{drinker='John'}(Likes) \bowtie Serves$$

$\sigma_{drinker='John'}$ (Likes)	
drinker	**beer**
John	Hoegaerden
John	Westmalle

⋈

Serves	
bar	**beer**
Zur Laube	Duvel
Zur Laube	Westmalle
Zur Laube	Hoegaerden
Albrecht	Hoegaerden
Albrecht	Chouffe
Kugel	Westmalle

➔

$\sigma_{drinker='John'}$ (Likes) ⋈ Serves		
drinker	**bar**	**beer**
John	Zur Laube	Hoegaerden
John	Albrecht	Hoegaerden
John	Kugel	Westmalle

Talk Outline

1. Primer on databases
2. Representing tuples, relations in DNA
3. Doing relational algebra by DNA computing
4. DNAQL, the language
5. DNA complexes: the DNAQL data model
6. Typechecking
7. Expressive power of DNAQL

Use of DNA codewords

- 4-letter alphabet is a bit limiting
- Can use larger alphabet
  - Encode different letters by noninteracting DNA strands
  - library of DNA codewords

The alphabet we use

- Bits 0 and 1
  - Data entries in tuples: strings of ℓ bits
- Position markers: ϕ_1, …, ϕ_ℓ
- Attributes
- Tags: $\#_1$, $\#_2$, …, $\#_9$
  - Used for punctuation, marking, splitting

Tuples as DNA strands

- Tuple:

A	B
0101	1000

$$\#_2 A \#_3 \phi_1 0 \phi_2 1 \phi_3 0 \phi_4 1 \#_4 \#_2 B \#_3 \phi_1 1 \phi_2 0 \phi_3 0 \phi_4 0 \#_4$$

5’ → 3’

- Relation: set of DNA strings
- Content of a test tube

Talk Outline

1. Primer on database
2. Representing tuples, relations in DNA
3. Doing relational algebra by DNA computing
4. DNAQL, the language
5. DNA complexes: the DNAQL data model
6. Typechecking
7. Expressive power of DNAQL

Selection by affinity purification

- “Retrieve all tuples from test tube where some bit is 0”
- Perform affinity purification
- Probe: complementary codeword for 0

Abstract DNA operations

- Abstract protocol voor affinity purification:

1. Insert probe
2. Hybridize
3. Flush: wash away tuples that did not stick
4. Cleanup: recover remaining tuples

$\sigma_{\text{some bit is 0}}(R) =$

$\text{cleanup}(\text{flush}(\text{hybridize}(R \cup \text{immob}(0'))))$

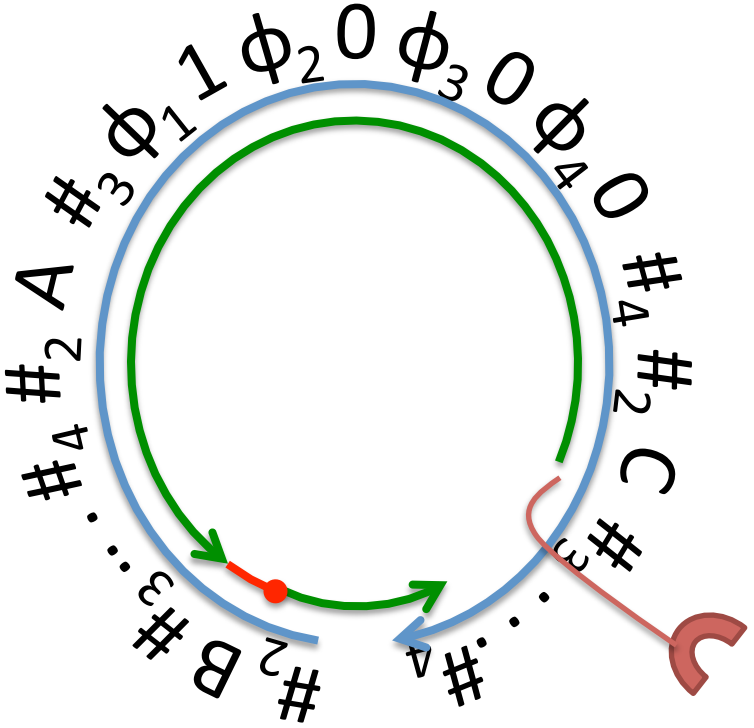
More selection

- So far we have done $\sigma_{\text{some bit is 0}}(R)$
- More interesting is $\sigma_{\text{drinker='John'}}(\text{Likes})$
- Let's do $\sigma_{\text{3rd bit of A=0}}(R)$
- Assume R has attributes B, A, C

$$\#_2\, B\, \#_3 \ldots \#_4\, \#_2\, A\, \#_3\, \phi_1\, 1\, \phi_2\, 0\, \phi_3\, 0\, \phi_4\, 0\, \#_4\, \#_2\, C\, \#_3 \ldots \#_4$$

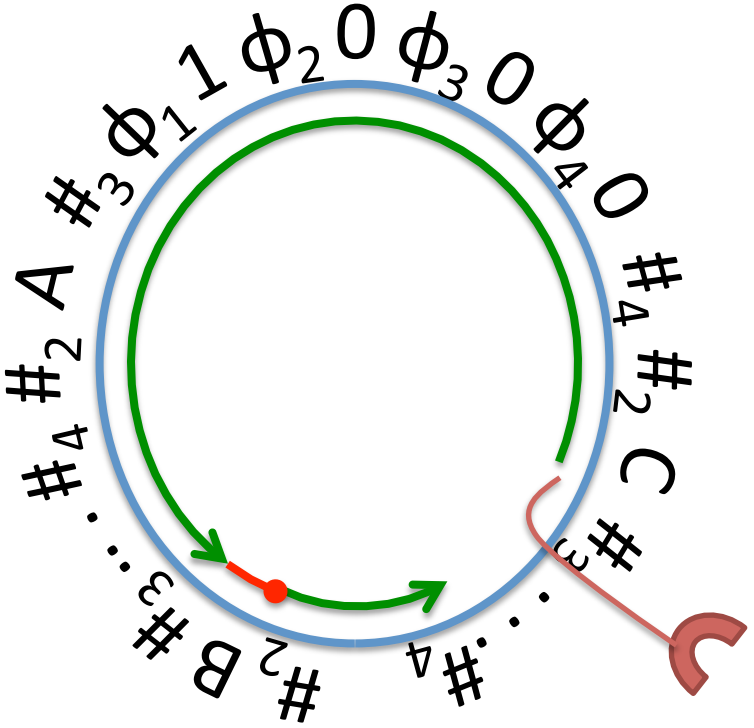
Circularize

1. block B (use ddB’)
2. polymerase (primer C’)
3. immobilize (probe $\#_3$’)
4. use bridge $\#_2$’$\#_4$’
5. ligate



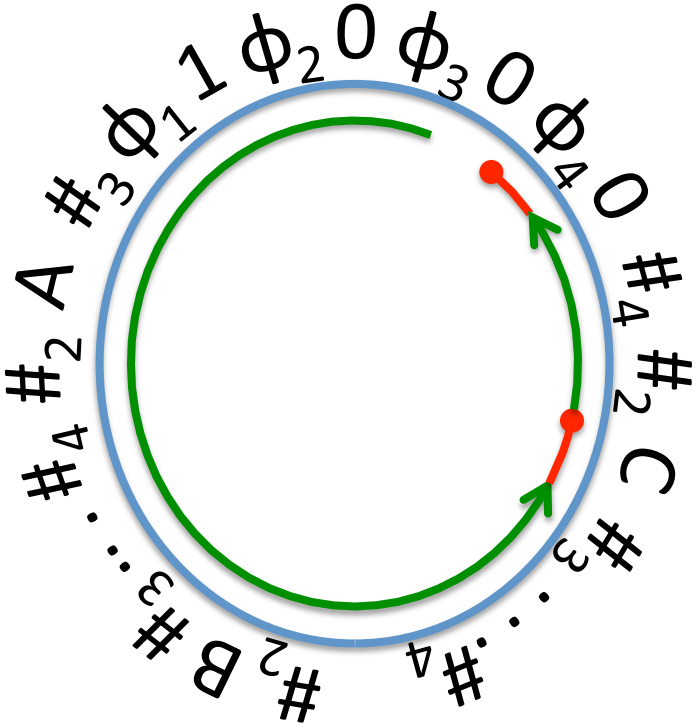
Circularize

1. block B (use ddB’)
2. polymerase (primer C’)
3. immobilize (probe $\#_3$’)
4. use bridge $\#_2$’$\#_4$’
5. ligate



$\sigma_{\text{3rd bit of A=0}}(R)$

1. block C
2. polymerase (primer ϕ_3')
3. block ϕ_4
4. polymerase with $\#_2$'
5. probe for 0'



Join (⋈), Cartesian product (×)

- R × S combines each tuple from R with each tuple from S

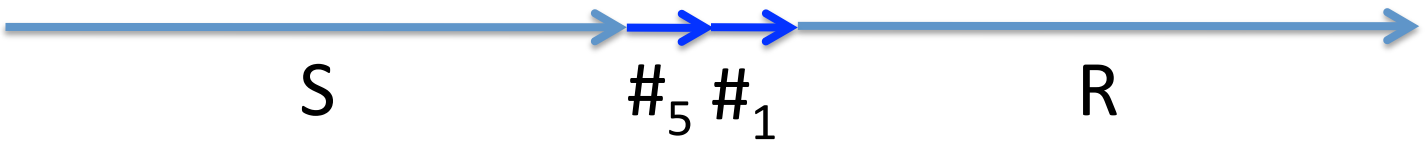
$$R \times S = \{ t_1 t_2 : t_1 \text{ in } R \text{ and } t_2 \text{ in } S \}$$

Algorithm for R × S

1. append $\#_5$ to every tuple of S (use bridge $\#_5$'$\#_4$' and ligate)
2. prepend $\#_1$ to every tuple of R (use bridge $\#_2$'$\#_1$' and ligate)

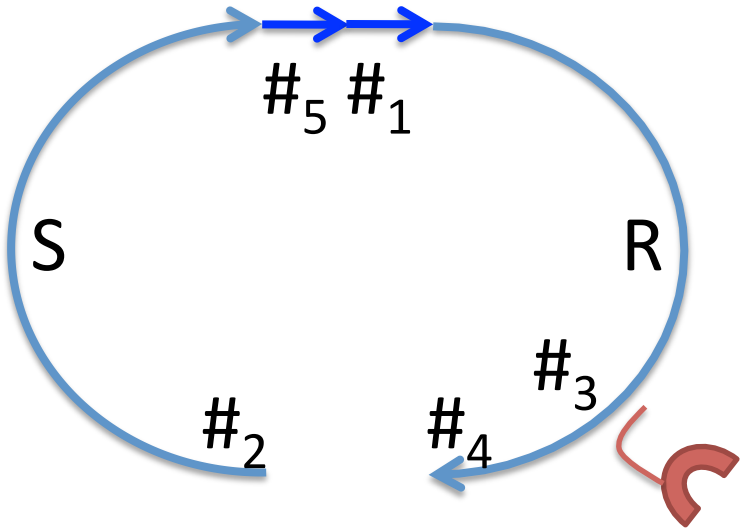
Algorithm for R × S

1. append $\#_5$ to every tuple of S
2. prepend $\#_1$ to every tuple of R
3. concatenate (use bridge $\#_1$'$\#_5$')



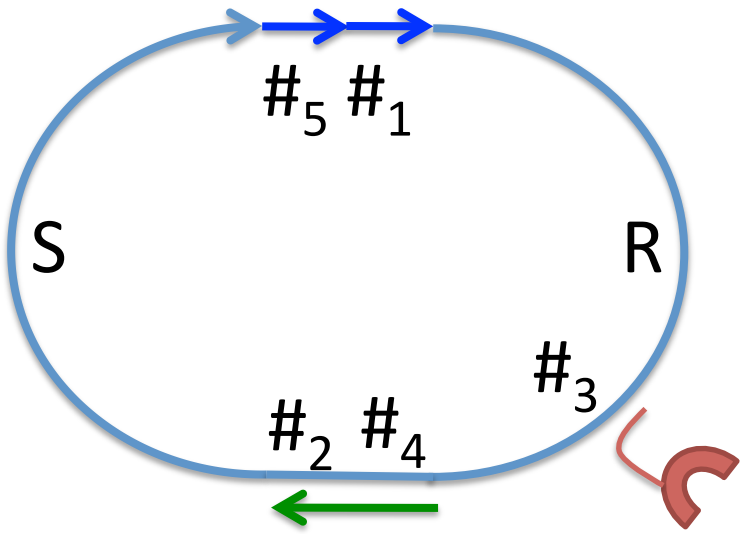
Algorithm for R × S

1. append $\#_5$ to every tuple of S
2. prepend $\#_1$ to every tuple of R
3. concatenate
4. immobilize



Algorithm for R × S

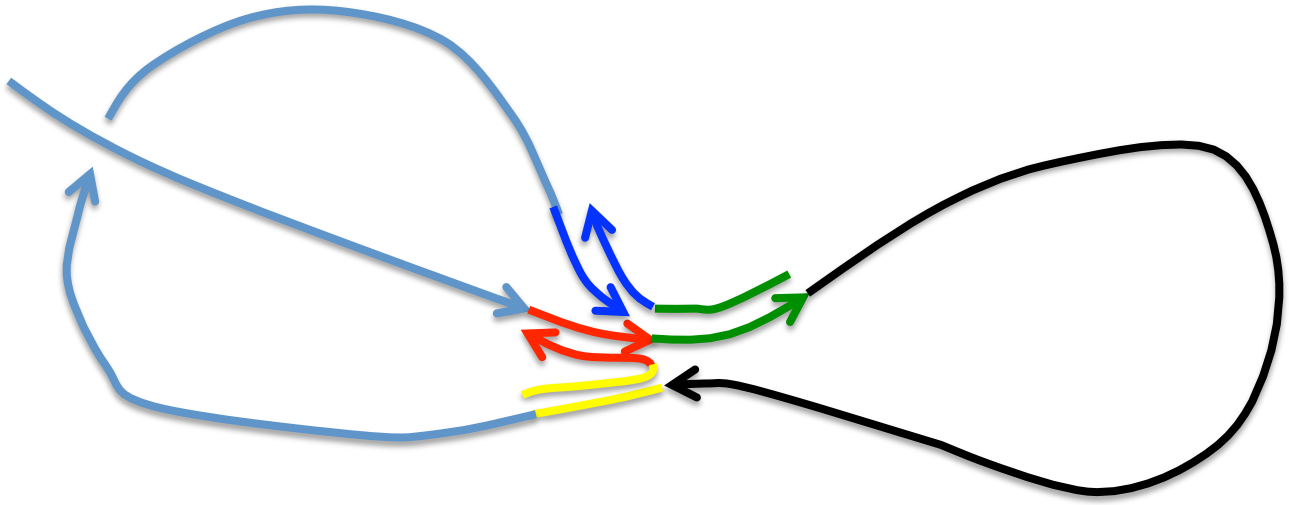
1. append $\#_5$ to every tuple of S
2. prepend $\#_1$ to every tuple of R
3. concatenate
4. immobilize
5. circularize
6. cleave at $\#_5$ and $\#_1$



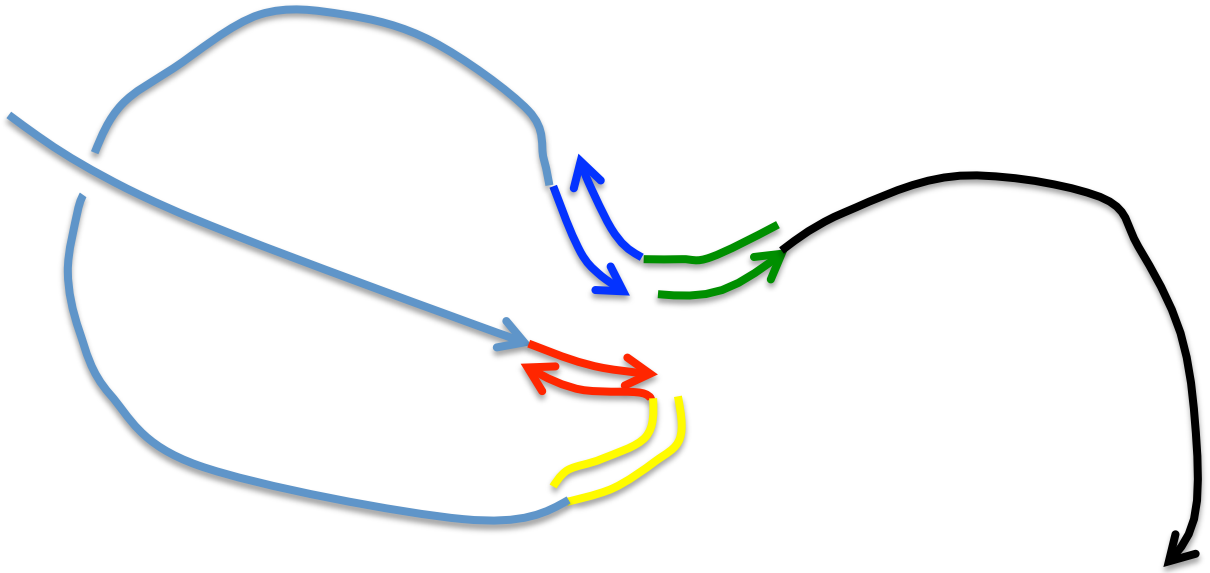
Other relational algebra operations

- Projection, renaming
  - similar methods
- Set difference R − S
  - subtractive hybridization
  - tuples in R and S have same length

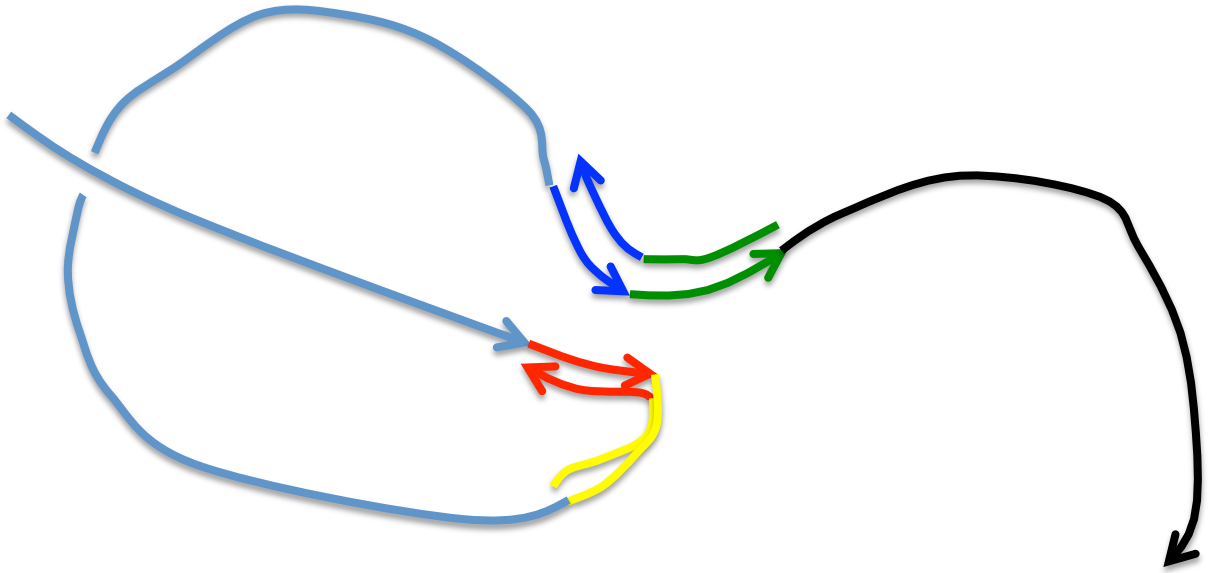
Shuffling attributes by double bridging



Shuffling attributes by double bridging



Shuffling attributes by double bridging



Abstract DNA operations

- Test-tube variables
- Probes
- Length-two bridges
- Union
- Difference
- For-loop
- Hybridize
- Ligate
- Flush
- Cleanup
- Cleave
- Block
- Polymerase, primer

For-loop

- DNAQL program for $\boldsymbol{\sigma}_{A=B}(R)$:

$$\texttt{for } s := r \texttt{ iter } i \texttt{ do}$$

$$\boldsymbol{\sigma}_{B=_i 0}\, \boldsymbol{\sigma}_{A=_i 0}(s) \cup \boldsymbol{\sigma}_{B=_i 1}\, \boldsymbol{\sigma}_{A=_i 1}(s)$$

DNA Query Language

$$
\begin{array}{rcl}
\langle expression \rangle & ::= & \langle complexvar \rangle \mid \langle foreach \rangle \mid \langle if \rangle \mid \langle let \rangle \mid \langle operator \rangle \mid \langle constant \rangle \\
\langle foreach \rangle & ::= & \texttt{for}\ \langle complexvar \rangle := \langle expression \rangle\ \texttt{iter}\ \langle counter \rangle\ \texttt{do}\ \langle expression \rangle \\
\langle if \rangle & ::= & \texttt{if empty}(\langle complexvar \rangle)\ \texttt{then}\ \langle expression \rangle\ \texttt{else}\ \langle expression \rangle \\
\langle let \rangle & ::= & \texttt{let}\ x := \langle expression \rangle\ \texttt{in}\ \langle expression \rangle \\
\langle operator \rangle & ::= & ((\langle expression \rangle) \cup (\langle expression \rangle)) \mid ((\langle expression \rangle) - (\langle expression \rangle)) \\
& \mid & \texttt{hybridize}(\langle expression \rangle) \mid \texttt{ligate}(\langle expression \rangle) \\
& \mid & \texttt{flush}(\langle expression \rangle) \mid \texttt{split}(\langle expression \rangle, \langle splitpoint \rangle) \\
& \mid & \texttt{block}(\langle expression \rangle, \Sigma - \Lambda) \mid \texttt{blockfrom}(\langle expression \rangle, \Sigma - \Lambda) \\
& \mid & \texttt{blockexcept}(\langle expression \rangle, \langle counter \rangle) \mid \texttt{cleanup}(\langle expression \rangle) \\
\langle constant \rangle & ::= & \Sigma^+ \mid \left(\overline{\Sigma} - \overline{\Lambda}\right)\left(\overline{\Sigma} - \overline{\Lambda}\right) \mid \texttt{immob}(\overline{\Sigma}) \mid \texttt{empty} \\
\langle splitpoint \rangle & ::= & \#_2 \mid \#_3 \mid \#_4 \mid \#_6 \mid \#_8
\end{array}
$$

Fig. 5. Syntax of DNAQL.

"the relational algebra for DNA"

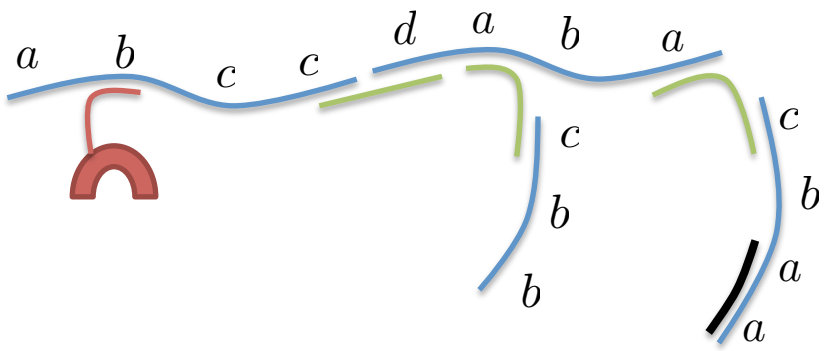
Talk Outline

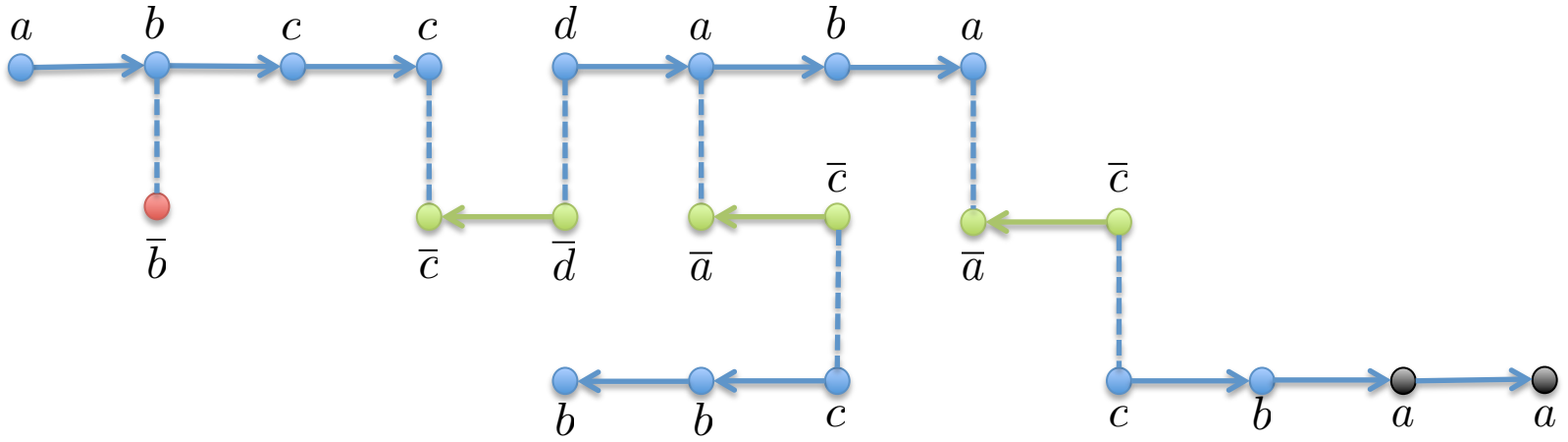
1. Primer on databases
2. Representing tuples, relations in DNA
3. Doing relational algebra by DNA computing
4. DNAQL, the language
5. DNA complexes: the DNAQL data model
6. Typechecking
7. Expressive power of DNAQL

Complexes

- Relation in DNA: set of DNA strings
- During execution of DNAQL program, more complex structures are formed
- Complexes formalized as directed graph
- Data model for DNAQL

DNA complex as a graph structure





Types

- If complexes are the “instances” in our data model, what are the “schemes”?
- Approach:
  - All data values are carried by strings of value bits
  - All other nodes are for structuring

➔ Type of a complex:

  - Replace all value strings by wildcard ‘*’

Type of a relation

relation	type
$\#_2A\#_3\underline{0011}\#_4\#_2B\#_3\underline{1100}\#_4$	
$\#_2A\#_3\underline{0001}\#_4\#_2B\#_3\underline{1101}\#_4$	
$\#_2A\#_3\underline{1011}\#_4\#_2B\#_3\underline{1100}\#_4$	$\#_2A\#_3{*}\#_4\#_2B\#_3{*}\#_4$
$\#_2A\#_3\underline{0011}\#_4\#_2B\#_3\underline{1111}\#_4$	
$\#_2A\#_3\underline{0000}\#_4\#_2B\#_3\underline{1111}\#_4$	

Talk Outline

1. Primer on databases
2. Representing tuples, relations in DNA
3. Doing relational algebra by DNA computing
4. DNAQL, the language
5. DNA complexes: the DNAQL data model
6. Typechecking
7. Expressive power of DNAQL

Well-definedness of DNAQL operations

- Implementability by biotechnological operations imposes some preconditions
- Always well-defined:
  - Union
  - Ligate
  - Split
  - Cleanup

Well-definedness conditions

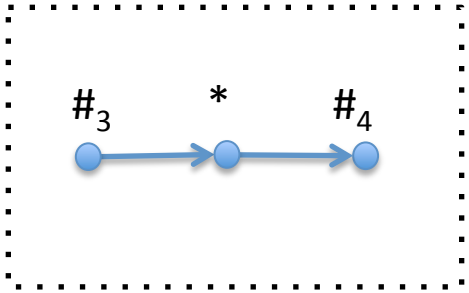
- Difference:
  - single strands only, all same length
- Blocking:
  - complex must be hybridized
- Hybridize:
  - termination (no chain reactions)
  - can be statically characterized in terms of absence of certain alternating cycles

Typechecking and inference

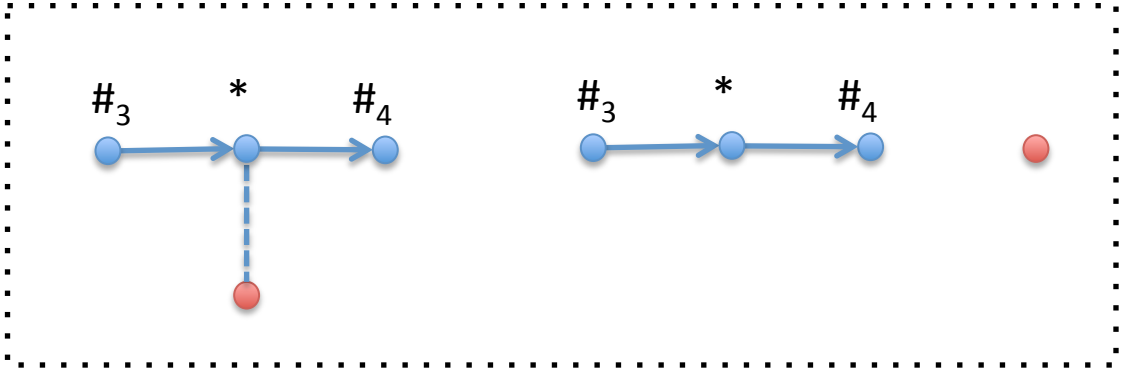
- Check well-definedness condition for operation statically, based on given input types
- Infer type for output, so that next operation can be typechecked

Type inference example

- e(x) = hybridize(x ∪ immob(ā))
- If x : S then e(x) : T



type S



type T

Typechecking Cleanup

- Input: any complex (always well-defined)
- Output: denature, remove all stickers, probes, keep only longest strands
- Gel electrophoresis

Typechecking Cleanup

- Consider type S = A*A*A ∪ AA*AA
- "Dimension" of a complex:
  - Number of value bits used for data values
  - Like word length in a digital computer
- Suppose dimension = d
  - Strands of type A*A*A have length 2d+3
  - Strands of type AA*AA length 4+d
  - 4+d < 2d+3 for all d

➔ If x : S then Cleanup(x) : A*A*A

Type inference algorithm

- Given input types for program:
  - Decides if “well-typed”
  - If so, computes result type
- Soundness: Well-typed programs always succeed on inputs of given type
  - Output guaranteed to be of computed result type
- Maximality: Converse to soundness
  - Only for individual operations
- Tightness

Talk Outline

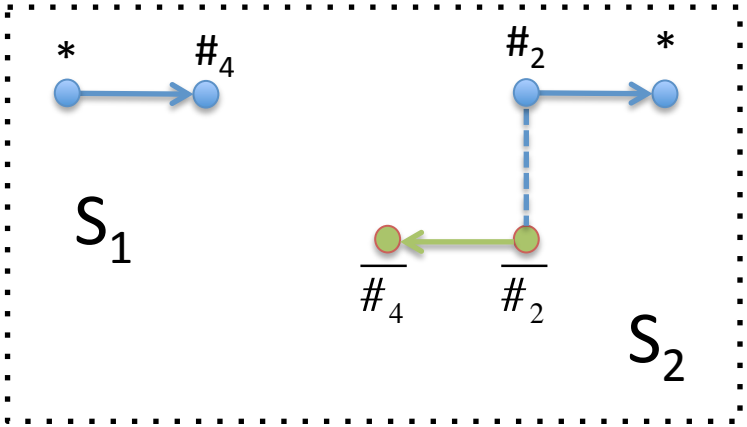
1. Primer on databases
2. Representing tuples, relations in DNA
3. Doing relational algebra by DNA computing
4. DNAQL, the language
5. DNA complexes: the DNAQL data model
6. Typechecking
7. Expressive power of DNAQL

Expressive power

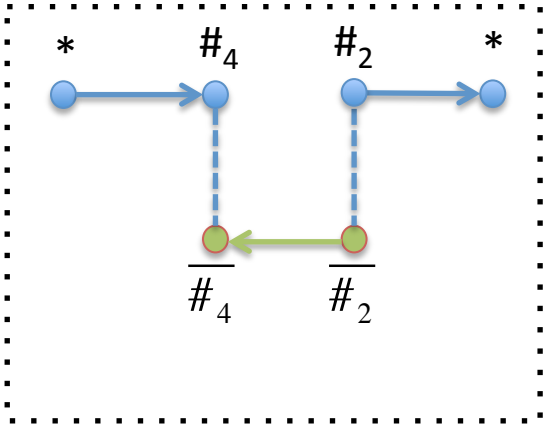
- We have seen that every relational algebra computation can be expressed by a DNAQL program
- Converse Theorem: DNA complexes can be simulated by relational databases, and DNAQL programs by relational algebra computations.

DNAQL to relational algebra

Type S



Type T



- If x : S then Hybridize(x) : T
- Store values in components of type S_1 in a relation R_1, similar for S_2
- Then pairs of values in components of Hybridize(x) can be computed R_1 x R_2
- Hybridization = Cartesian product!

Summary

- DNA computing develops algorithms for data represented in DNA
- Novel application area for biotechnology
- We have tried to find the equivalent of relational data model, relational algebra in the world of DNA computing
- Resulting DNAQL data model can stand on itself

Outlook

- Experiments?
- Simulation?
- Reliability?
- Self-assembly models of DNA computing
  - strand displacement

- References:
  - alpha.uhasselt.be/jan.vandenbussche