Robust and Universal Reversible Machines & High-Level Programming Languages in a Recombinase DNA System

A Forthcoming Proposal

- Principal Investigators:
  - Junghuei Chen (Chem., U. Del.)
  - Michael Frank (CISE, U. Fla.)
  - Harvey Rubin (Med., U. Penn.)
  - David Wood (CIS, U. Del.)

Logical Architecture Summary

Universal computing

- Interpreter for PsiLisP high-level language.
- Baker's reversible pointer automaton.
- Baker's *linear lisp* cons cell data structures?

Finite state machines

- Arbitrary finite reversible logic circuits.
- Fredkin's reversible boolean logic gates.

Abstract Logical Primitives

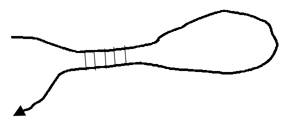
- Data-dependent conditional swap operations.
- Unconditional swap operations.
- Symbol (& binary tree?) data types.
- Indefinitely many named storage registers.

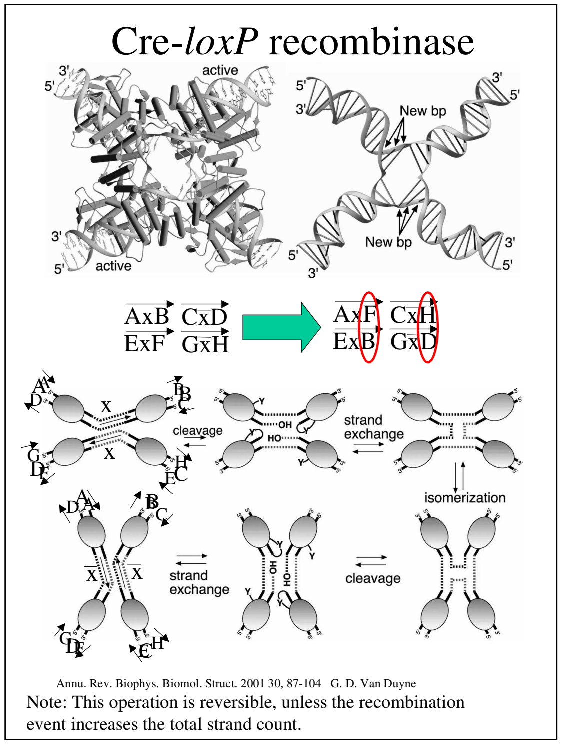
Biochemical Tools

- Selective site blocking w. oligo binding
- Site-specific recombination using Cre-*loxP*
- Self-hybridization & 2ndary structure formation
- Long custom ssDNAs using vast sequence libs.

Biochemical Tools

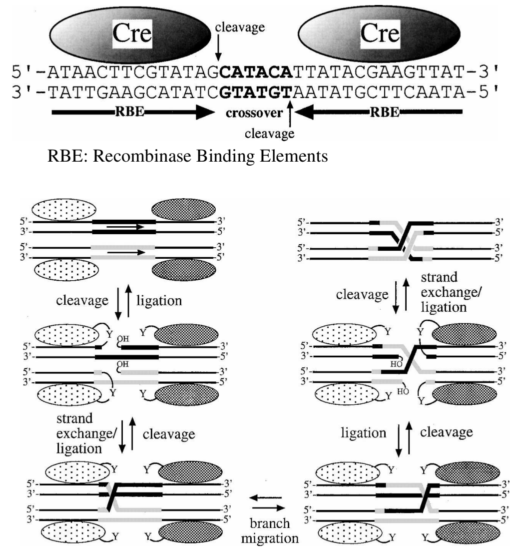
- Large custom sequence library
  - Funded project, shown yesterday
  - Goal: Many, distinct oligo sequences, no side reactions
- Initial program state: a custom ssDNA string of “symbols”
  - Intentional complementary symbols can form dsDNA in certain selected regions





Same questions here.

This is apparently a different view of recombination, that replaces the strings AxB and CxD with AxD and CxB. This is functionally equivalent to the previous one.



Annu. Rev. Biophys. Biomol. Struct. 2001 30, 87-104 G. D. Van Duyne

This makes no sense to me. If the black and the gray strands are different sequences, then how can gray be paired to black in the right-hand drawings?

If they are the same sequence, then I guess what we have accomplished is simply to swap the right ends of the two original double-strands with each other.

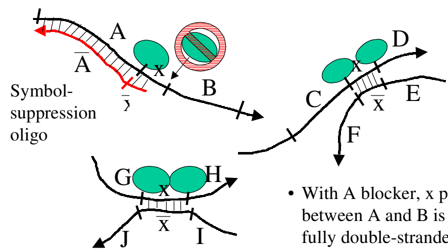
Let's describe this operation in terms of string manipulations. We start with two (double-stranded) strings $S_L b_1 x b_2 S_R$ and $T_L b_1 x b_2 T_R$, where b_1 and b_2 are (constant, predetermined) binding sequences, x is the crossover region, and S_L, S_R, T_L, and T_R are arbitrary and indefinitely-long strings to the left and to the right. After the crossover, we simply have $S_L b_1 x b_2 T_R$ and $T_L b_1 x b_2 S_R$. But it seems that this accomplishes absolutely *nothing* computationally, *unless there is some other operation that is affected by this change*. (In life, the change is "sensed" by having each new strands go off and potentially produce a new organism.)

Another important question: Do the light gray and dark gray recombinases recognize different sequences? How many available binding sequences do we have in our library?

Can the crossover region be self-complementary? Is it fixed by the enzyme, or can it be any double-stranded sequence? Is its length limited?

Selective Site Blocking w. Oligo Hybridization

Oligos (added to & removed from solution according to predefined program) selectively bind to & suppress crossover sites next to certain recognized symbols or symbol sequences.

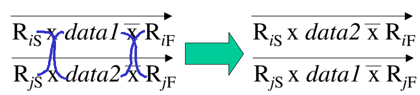


Without blocker, B could swap places with H, or B with D, or H with D.

- With A blocker, x point between A and B is not fully double-stranded.
- Full Cre-*loxP* assembly can't bind there.
- B is prevented from participating, because of its location (next to A).
- Only D & H can swap places!

Storage Registers

- May have indefinitely many registers R*i*: R0, R1, R2, …
  - Addessed by unique labels.
  - Number limited only by size of symbol-sequence library.
- Registers may hold data of any defined type
  - Atomic symbols: **0**, **1**, **null**, etc.
  - Composite data objects?
- Registers accessed via *exchange*
  - Suppress the *framing labels* of all regs but the 2 to be swapped



Note: Can implement an irreversible move operation by having a blocker that will bind to the source location once it is empty:

```
Ris x data X Rif
Rjs x null X Rjf
```

```
Ris x null X Rif
Rjs x data X Rjf
```

Data-Dependent Conditional Swap Operation

- Implements:
  If R_i = **c** then swap(R_{i-1}, R_{i+1})
- Blocker oligo recognizes **c** in central register & stops swap.

R_{1L} x *data1* $\overline{x}$ R_{2L} x *data2* $\overline{x}$ R_{3L} x *data3* $\overline{x}$ R_{3R}

x $\overline{R_{2L}}$ $\overline{x}$ $\overline{\mathbf{c}}$ x $\overline{R_{3L}}$

Caveats: (1) Must tune temperature, concentration, label lengths &/or # of base-pair mismatches for blocker oligos so that overall binding probability is sensitive to whether the controlling register 2 data region matches it. (2) Needs extra refinements to irreversibly lock in final state. (Work in progress.)

```
R1s x S1ls x null X S1lf X R1cl x ctl X R1cr x S1rs x null X S1rf X R1f
```

```
R1s x S1ls x dat1 X S1lf X R1cl x ctl X R1cr x S1rs x dat2 X S1rf X R1f
```

```
R1s x S1rs x dat2 X S1rf X R1cl x ctl X R1cr x S1ls x dat1 X S1lf X R1f
```

Have lock-in blocker made of PNA instead of DNA so it will displace the existing competing oligos.

```
R1s x S1rs x dat2 X S1rf X R1cl x ctl X R1cr x S1ls x dat1 X S1lf X R1f
R1s x s1ls          S1lf X R1cl          R1cr x S1rs x        X S1rf X R1f
```

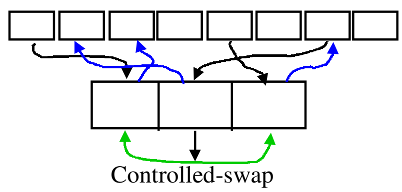
Now, have this set of blockers to suppress everything but the leftmost data region, to swap out the left data. Similarly for the rightmost data region, and the control region.

```
R1s x S1rs x null X S1rf X R1cl x nul X R1cr x S1ls x null X S1lf X R1f
      S1ls x
```

Finally, swap the shuttles back to their original position (unless they are still in their original positions already).

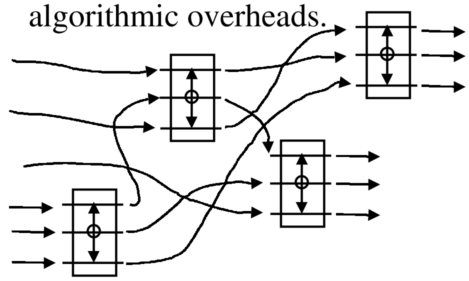
Fredkin Gates

- Registers contain bits
- Special controlled-swap gate
- Three-step process:
  - Move 3 inputs into gate
  - Do the controlled swap
  - Move 3 cells to output locations



Reversible Circuits

- Compose arbitrary reversible boolean circuits from Fredkin Gates.
- Build a reversible computer.
- Run any program, with some algorithmic overheads.



Logical Architecture Summary

Universal computing

- Interpreter for PsiLisP high-level language.
- Baker's reversible pointer automaton.
- Baker's *linear lisp* cons cell data structures?

Finite state machines

- Arbitrary finite reversible logic circuits.
- Fredkin's reversible boolean logic gates.

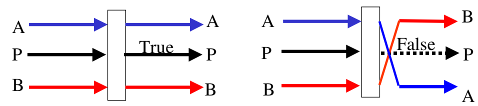
Abstract Logical Primitives

- Data-dependent conditional swap operations.
- Unconditional swap operations.
- Symbol (& binary tree?) data types.
- Indefinitely many named storage registers.

Biochemical Tools

- Selective site blocking w. oligo binding
- Site-specific recombination using Cre-*loxP*
- Self-hybridization & 2ndary structure formation
- Long custom ssDNAs using vast sequence libs.

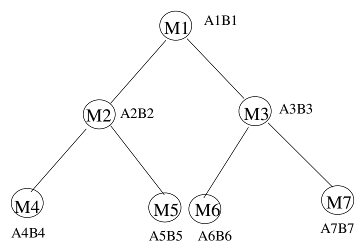
Conditional Exchange of Program Segments



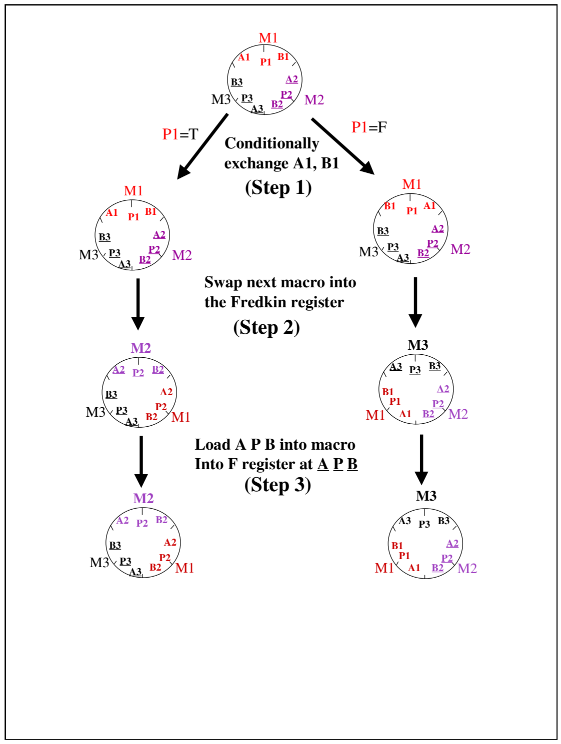
Macro (k, $\underline{A}$, $\underline{P}$, $\underline{B}$):

From $\underline{A}$, $\underline{P}$, $\underline{B}$, load A_k, P_k, B_k
If P_k=T
Then $A_kB_k \rightarrow A_kB_k$
And go to Macro m2k
Else $A_kB_k \rightarrow B_kA_k$
And go to Macro m2k+1

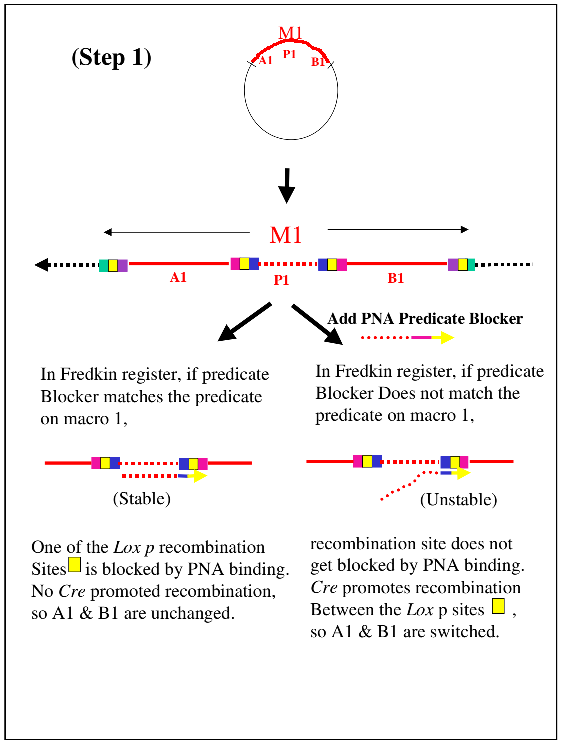
Example Program: Decision Tree:



By itself this seems a very weak computational model. Rather than trying to do something with this concept, I would prefer to throw it out entirely, take a step backwards, and start by considering what RBEs do, from an abstract perspective, and see if I can build my own computing model starting with them.



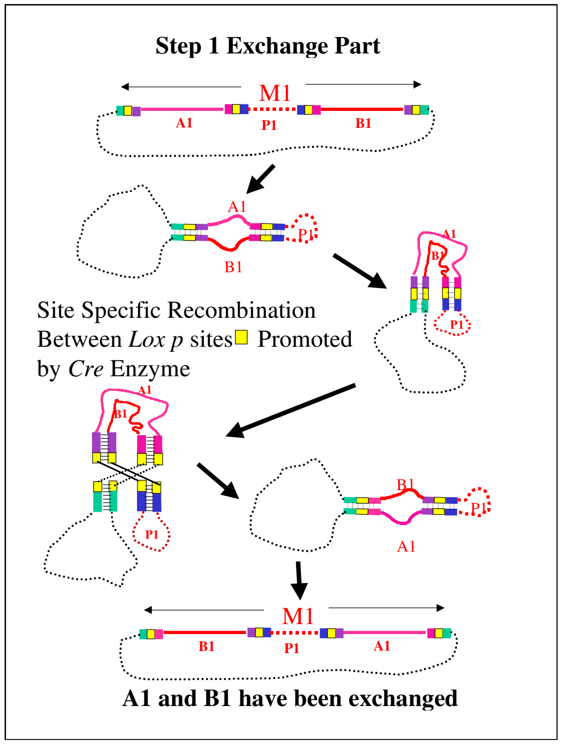
Again it seems that this model, even if implemented, isn't very powerful computationally.



What is the PNA predicate blocker binding to? The entire P1 strand? How much of the framing sequences are involved?

Apparently the plasmid here is single-stranded.

Effectively, it seems that a given predicate is either on or off depending on whether the blocker is present.



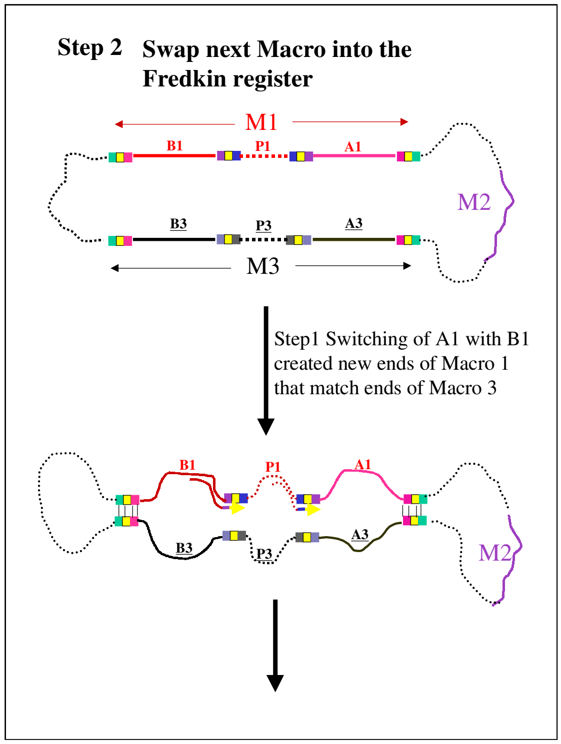
If I'm understanding correctly, what's happening here is that short sequences like gyp are binding to their (complementary) mates, strongly enough to forma crossover site (one for each double-stranded pair of yellow blocks). Then we cross over 2 crossover sites within the same molecule, turning $gx\underline{v\mathit{A}p}xb\mathit{P}b'x\underline{p'\mathit{B}v'}yg' \rightarrow gx\underline{p'\mathit{B}v'}xb\mathit{P}b'x\underline{v\mathit{A}p}yg'$ (underlined parts are swapped). x is now the recognition+crossover site.

This is OK but it seems that the only real action, computationally speaking, has been to change the immediate neighborhood of the Lox p sites. Ie., in the original (pre-exchange) state, both A and B had green-yellow-purple on one side, and pink-yellow-blue on the other. After the exchange, we have green-yellow-pink on one side, and purple-yellow-blue on the other.

The important effect of the "exchange" operation is not that it is exchanging A and B (since their relative position is irrelevant), but that it is *reversing the orientations* of both A and B relative to the sequences that frame them.

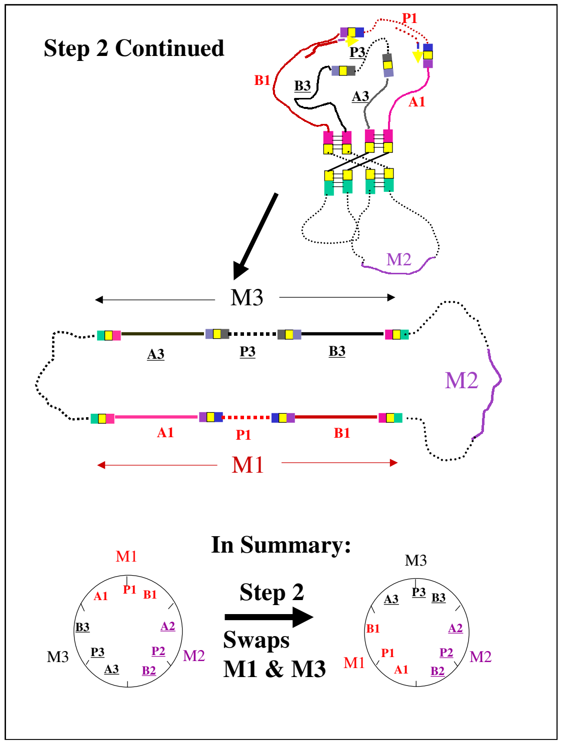
Both before and after the swap, A and B are both framed by yellow-green on one side, and yellow-blue on the other. A and B also both have one purple end, and one pink end. However, before the swap, the purple end meets the green side of the frame, and the pink end meets the blue side of the frame. After the swap, A and B are both rotated 180 degrees relative to their frames.

In effect, what this does is replace the GYPu & PiYB sequences that existed in the original situation with the GYPi and PuYB sequences that exist after. I don't see how the actual *contents* of A and B are relevant to anything.



OK, now the creation of the new GYP sequences causes them to match the existing GYP sequences framing macro 3, fine. Again, it seems all the action is in the creation of the new framing sequences, not in the contents of A or B.

In this figure, what stopped macro 3 from executing earlier?



Now, we do an unconditional swap of everything between those sequences, presumably using a different enzyme that recognizes the new yellow sites but not the old ones. Or, if it is the same enzyme, I guess the previous step could proceed backwards instead of this step going forwards.

Anyway, note that this time both ends of the frame are identical: We have green on both ends. So, reversing the orientation of the macros seems to do nothing! It swaps their positions also, but this seems irrelevant. Computationally, this operation seems useless.

Binary Tree Data Type

- f

$$R_{iL}\ x\ (\ x\ (\ x\ a\ \overline{x}, x\ b\ \overline{x}\ )\ \overline{x}, x\ c\ \overline{x}\ )\ x\ R_{iR}$$