DNA-Templated Computing

Bjarke Niemann Hansen
Alexei Mihalchuk

Daniel Merkle, Kim Skak Larsen, Christoph Flamm

Syddansk Universitet

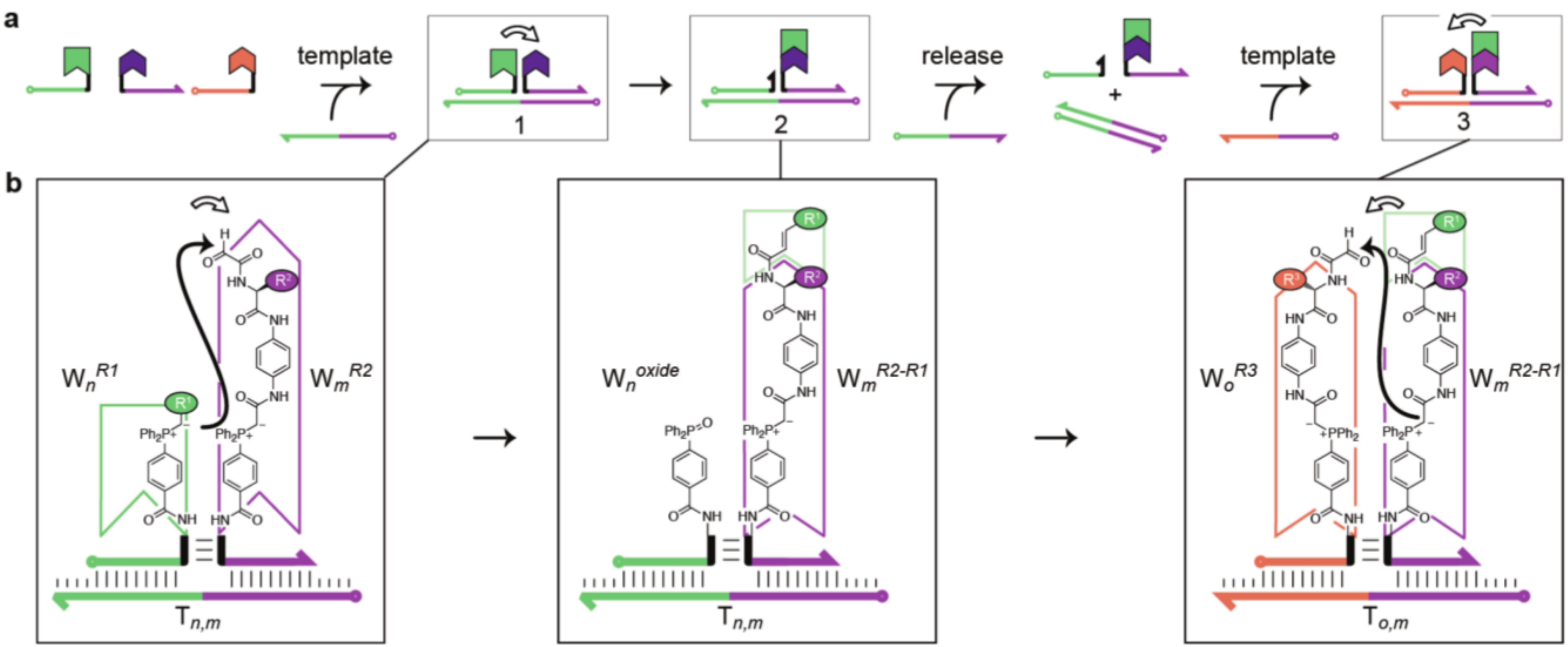
Our Thesis

- Given a synthesis plan, automatically infer an optimal DNA-templated program.

- Looking at **one-pot** synthesis from a *computer science* point of view.

Towards an Optimal DNA-Templated Molecular Assembler
Jakob L. Andersen, Christoph Flamm, Martin M. Hanczyc, Daniel Merkle

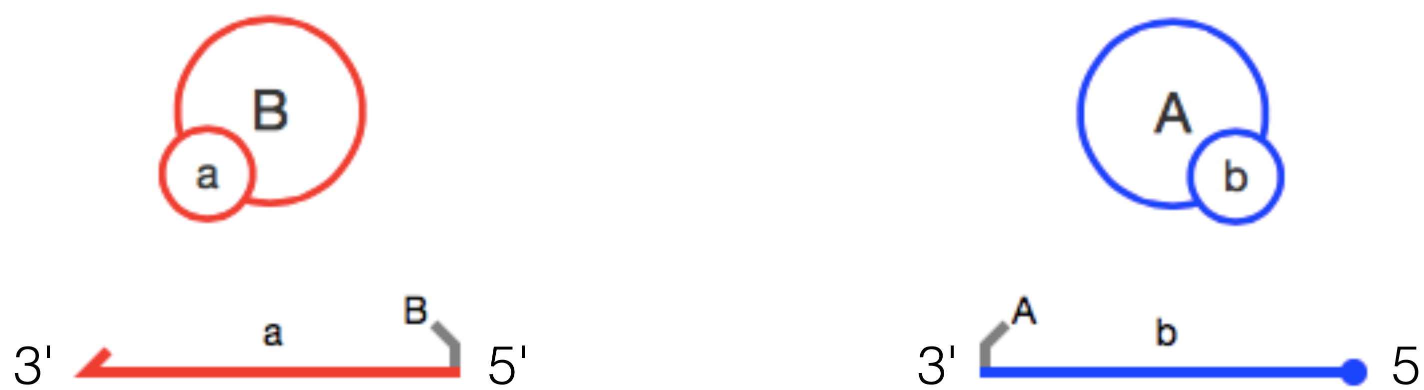
DNA-Templated Synthesis



Programmable One-Pot Multistep Organic Synthesis Using DNA Junctions - McKee et al. 2012

Basic Notation

- **Compound** (uppercase), **Tag** (lowercase)



- **Strand**: sequence of two tags

instruction

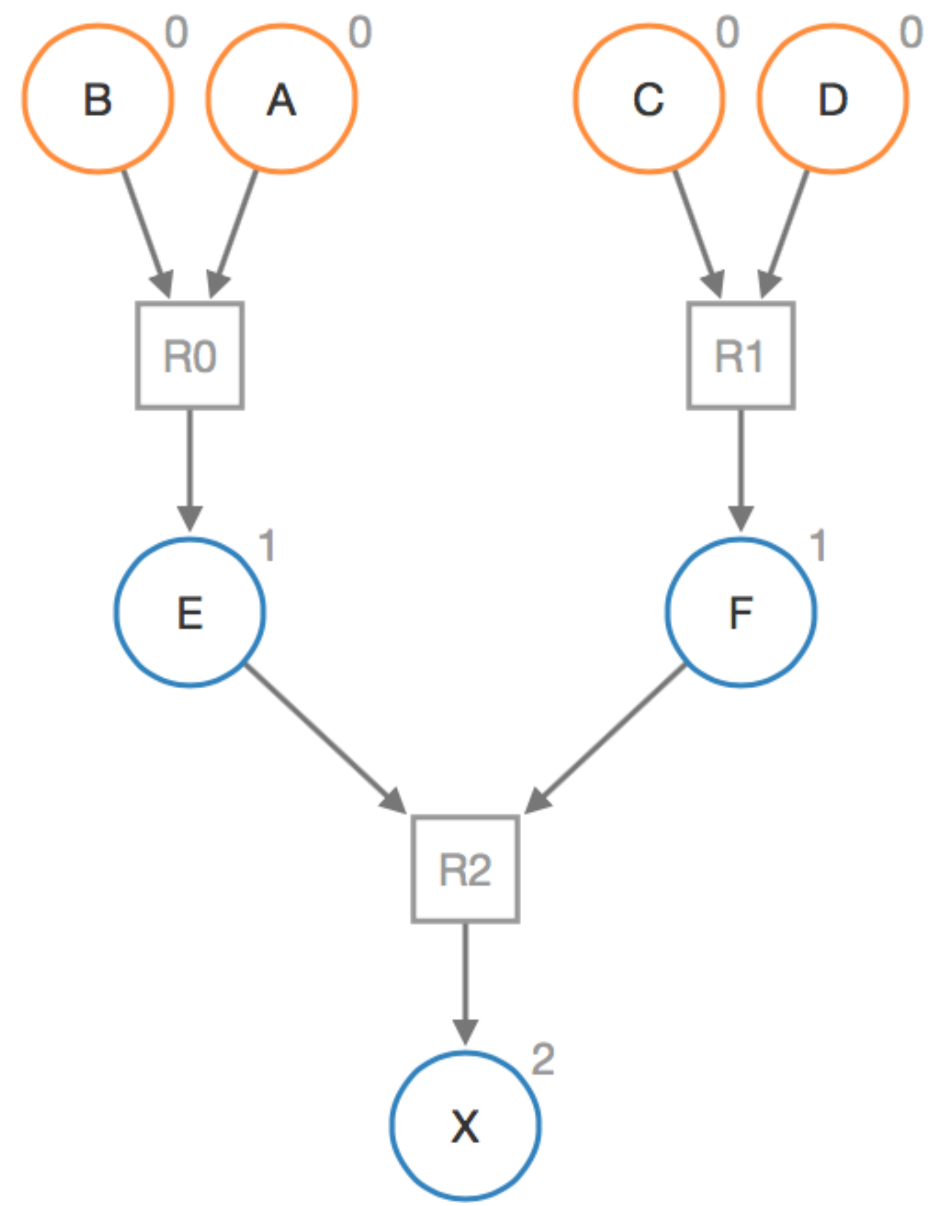
release

Synthesis Plan

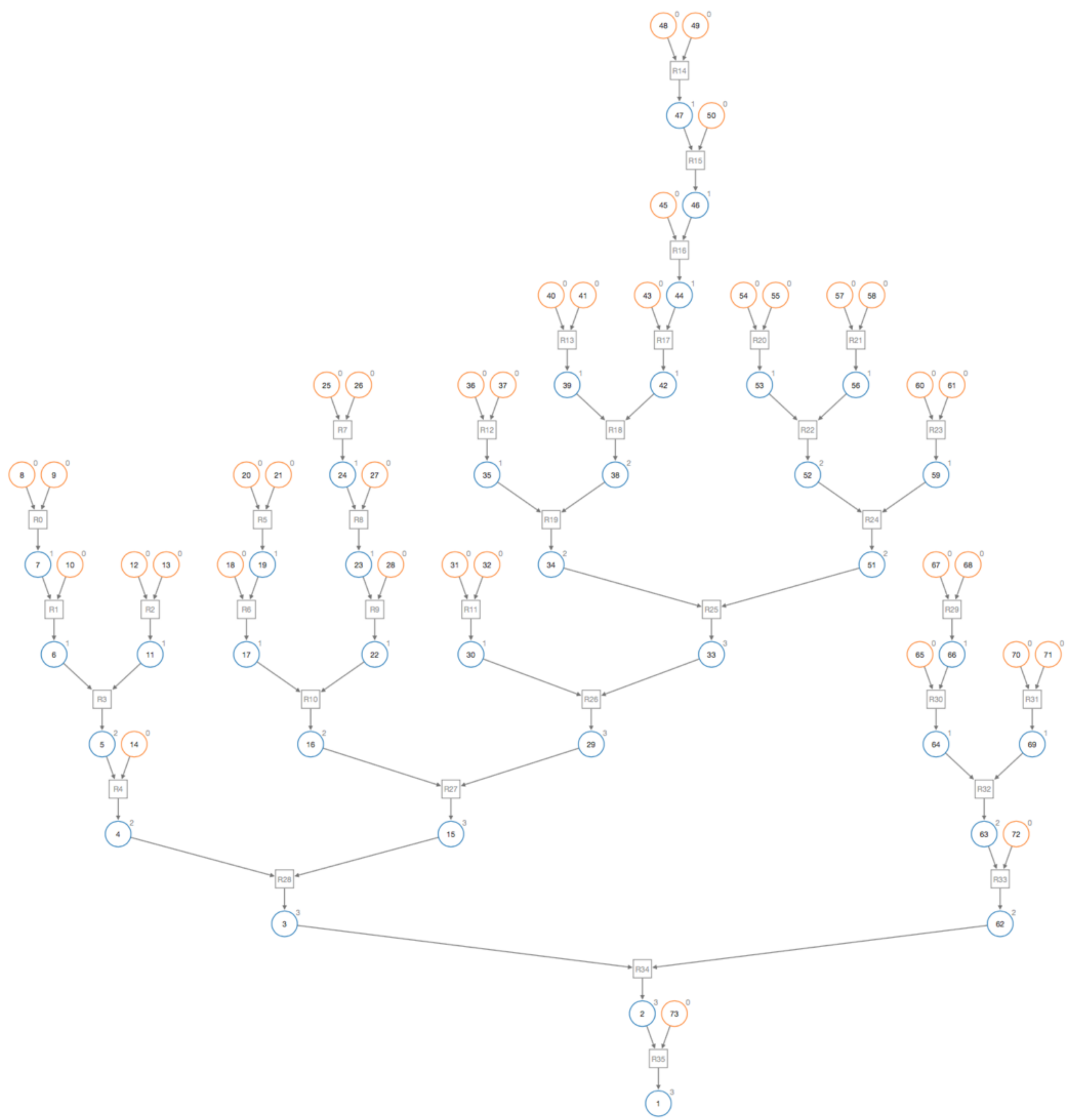
- A series of reactions that produces a goal compound.

- Assumed to be a binary tree.

```
B + A -> E
C + D -> F
E + F -> X
```

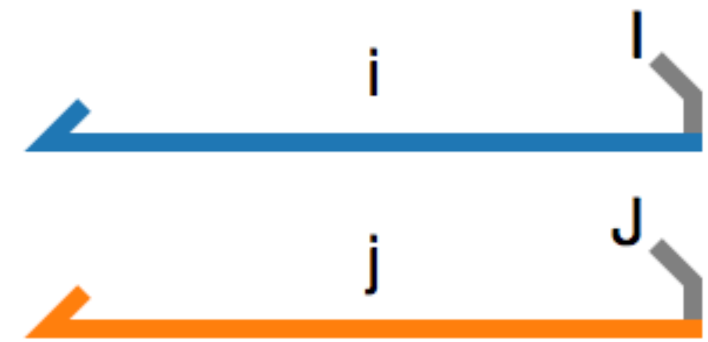


Synthesis Plan

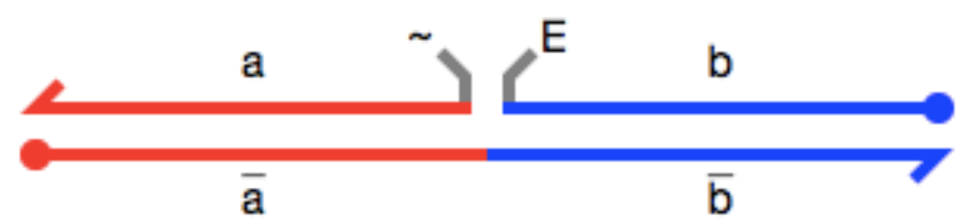


Basic Requirements

- Two compounds present in the pot at the same time must have a distinct tag.

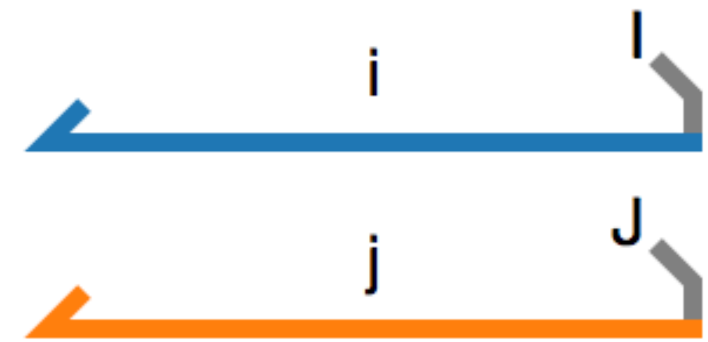


- Two strands present in the pot at the same time must differ in at least one of their tags.

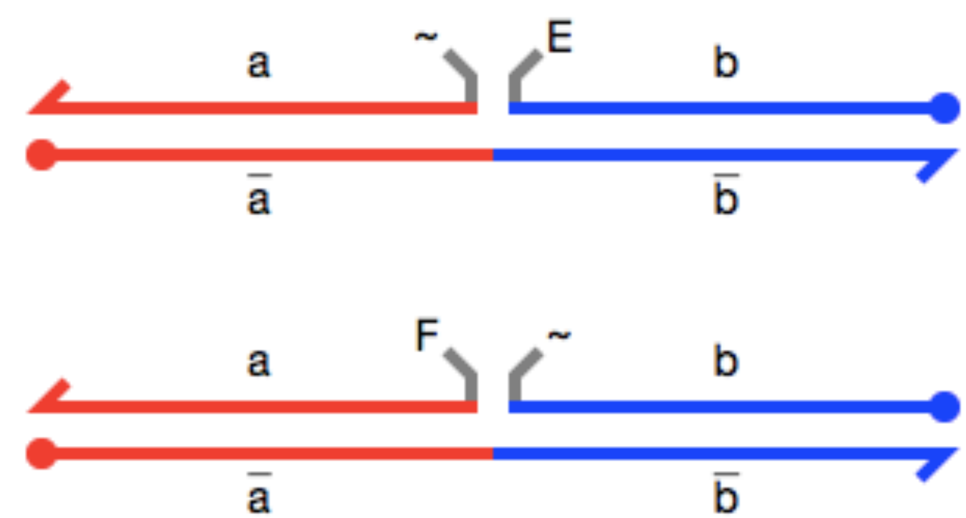


Basic Requirements

- Two compounds present in the pot at the same time must have a distinct tag.

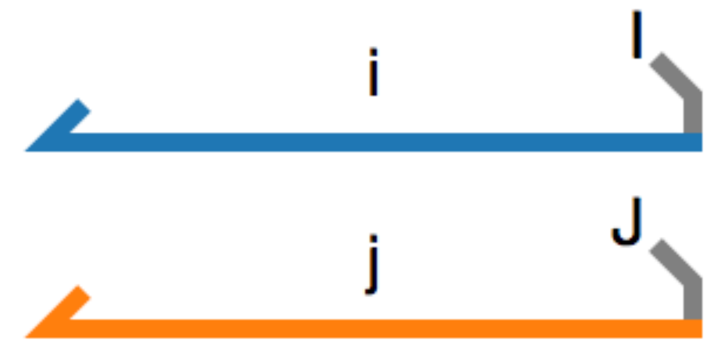


- Two strands present in the pot at the same time must differ in at least one of their tags.

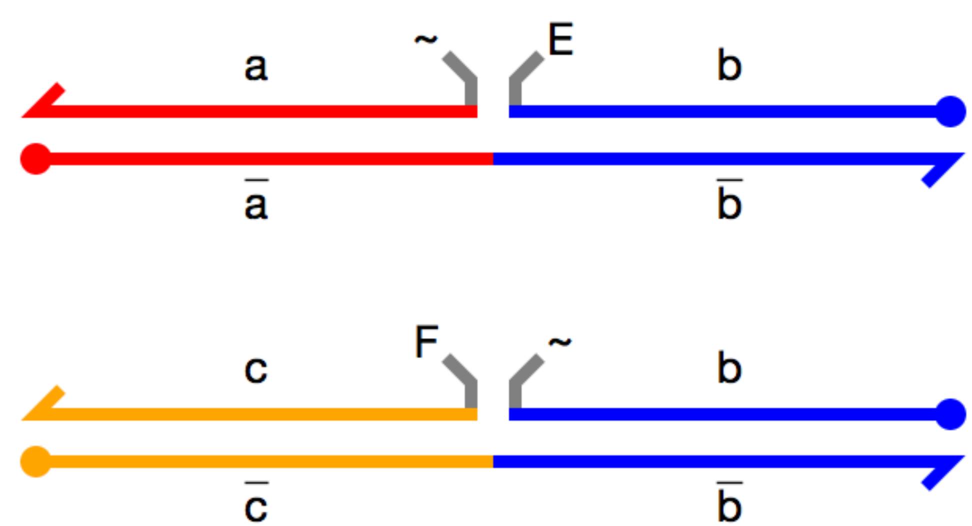


Basic Requirements

- Two compounds present in the pot at the same time must have a distinct tag.



- Two strands present in the pot at the same time must differ in at least one of their tags.



Motivating Example

Why is this hard?

- In which order to perform the reactions?
- Inheritance of tags and their position.
- Avoiding side products and unwanted bindings.
- Different optimization goals.

Optimization Goals

- *Program length*
  Fewer instructions but more distinct tags required.

- *No. of tags*
  Fewer tags, at the cost of a longer program.

- *No. of strands*
  Fewer distinct strands but not fewest tags.

Integer Linear Programming

- Inspired by liveness analysis from Compiler Theory.
- Given an arbitrary program, if two compounds or two strands are *alive* at the same time, they must be distinct.
- Solve such a program using an ILP model.

The Approach

- Enumerate all the ways to traverse a plan.
- Enumerate all the ways to construct a program.
- Determine the interference between the tags.
- Solve the interference constraints using ILP.

Interference

$$
\begin{array}{lll}
\text{minimize} & y & \\
\text{s.t.} & \sum_c x_{ic} = 1 & \forall i \in V \\
& x_{ic} + x_{jc} \leq 1 & \forall ij \in E, \forall c \in V \\
& x_{ic} + x_{kc} \leq 1 + z_{ijkl} & \forall ijkl \in S, \forall c \in V \\
& x_{jc} + x_{lc} \leq 1 + (1 - z_{ijkl}) & \forall ijkl \in S, \forall c \in V \\
& \sum_c c x_{ic} \leq y & \forall i \in V \\
& x_{ic} \in \{0, 1\} & \forall i, \forall c \in V \\
& y \geq 0 &
\end{array}
$$

Tags should be unique.

If two strands are alive at the same time, they should be unique.

Example

Integer Linear Programming

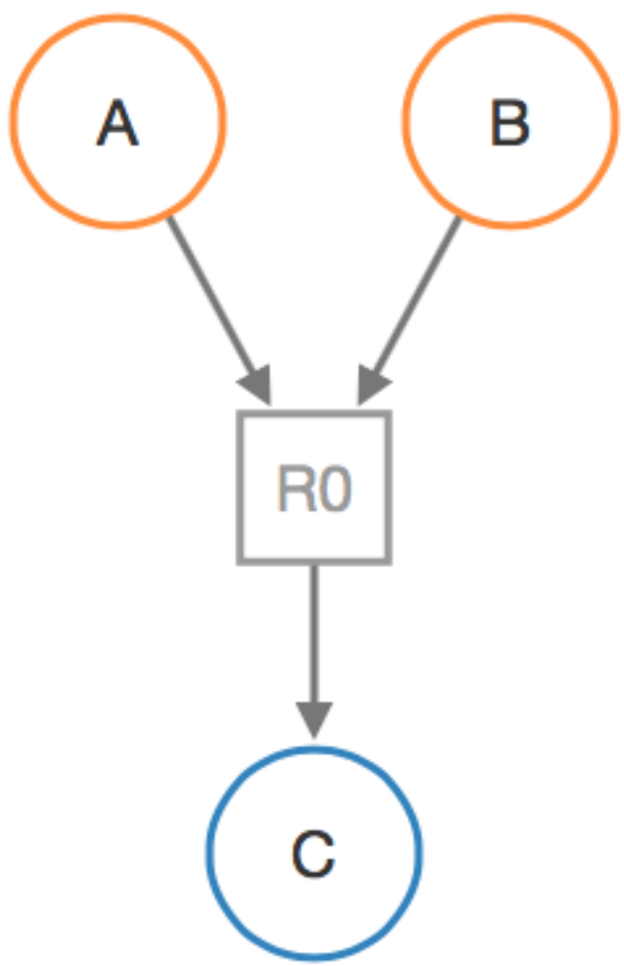
- An easy way to generate all (or a subset of) optimal programs for a synthesis plan.
- However, runs in exponential time and quickly becomes impractical.

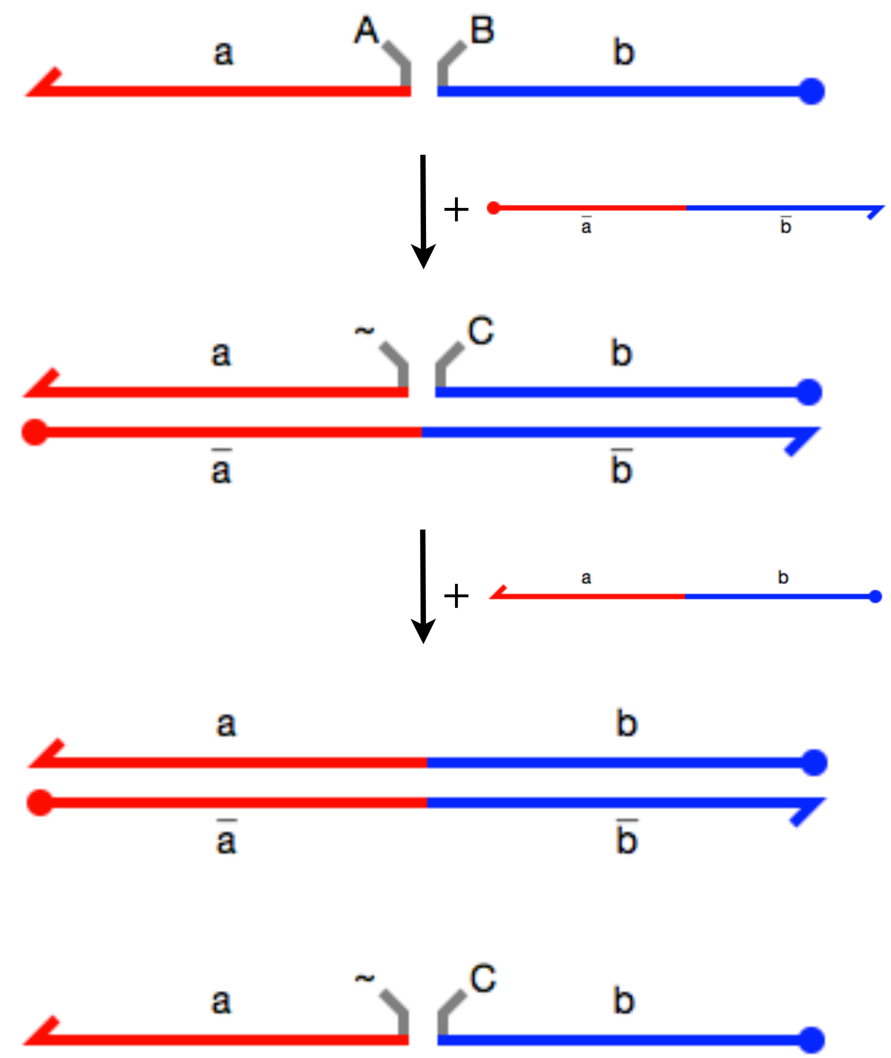
Tags on Compounds	Atomic	Toeholds	Fewest Tags	Shortest Length
2	Yes	No	Polynomial	Linear
2	Yes	Yes	Polynomial	Linear
2	No	No	Not applicable	Not applicable
2	No	Yes	Not applicable	Not applicable
Arbitrary	Yes	No	Polynomial	Linear
Arbitrary	Yes	Yes	*Exponential*	Linear
Arbitrary	No	No	Exponential	Linear
Arbitrary	No	Yes	Exponential	Linear
k	Yes	No	Polynomial	*Exponential*
k	Yes	Yes	*Exponential*	*Exponential*
k	No	No	Exponential	*Exponential*
k	No	Yes	Exponential	*Exponential*

Two Tags on Compounds

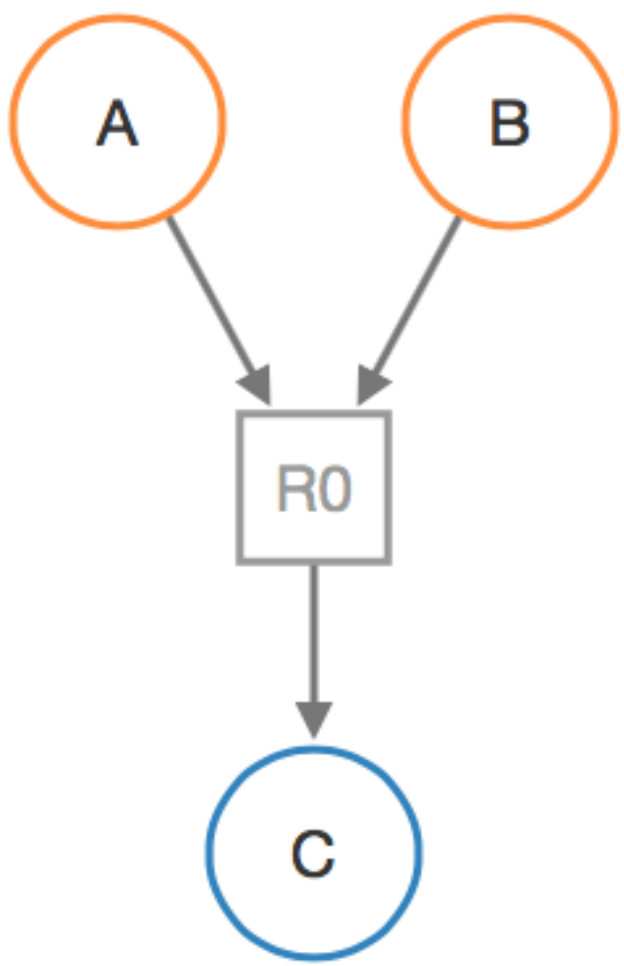
- Tagging compounds is expensive!
- Reuse the same two tags (*a* and *b*) for all the compounds.

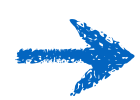
Atomicity

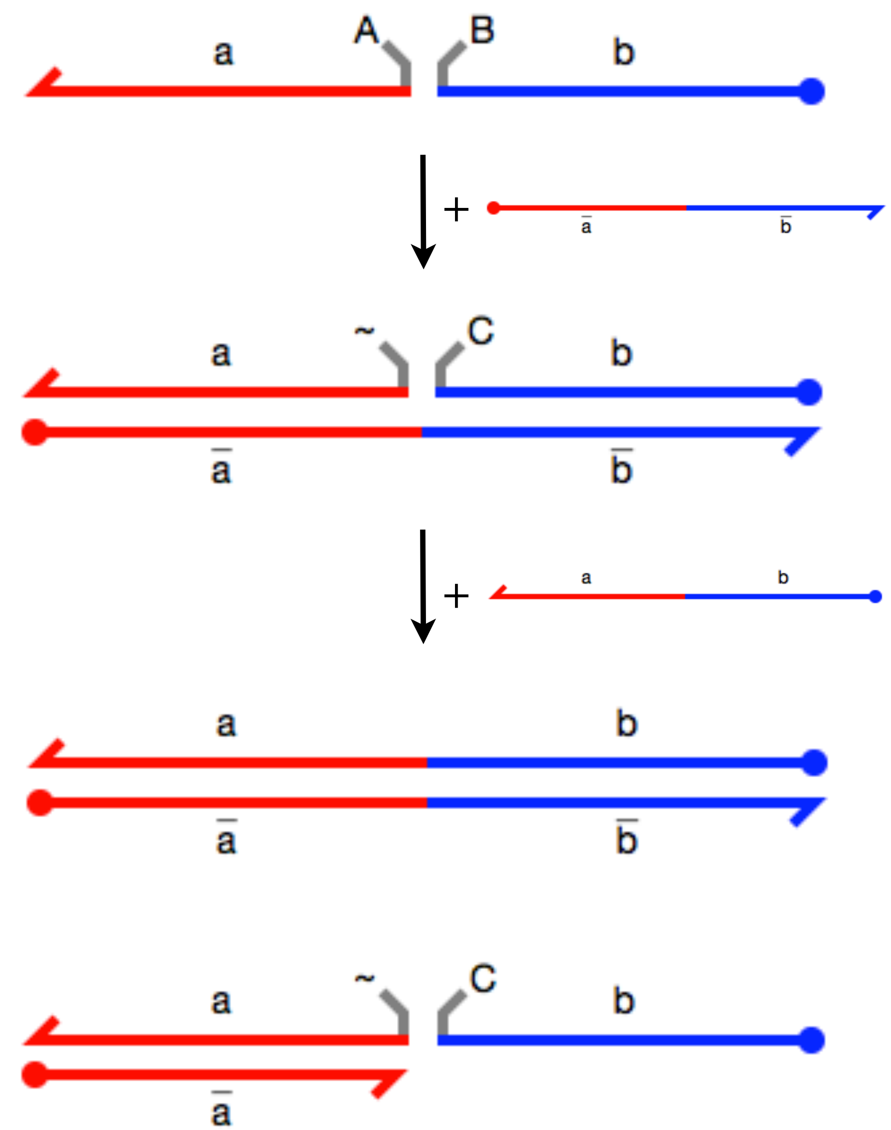




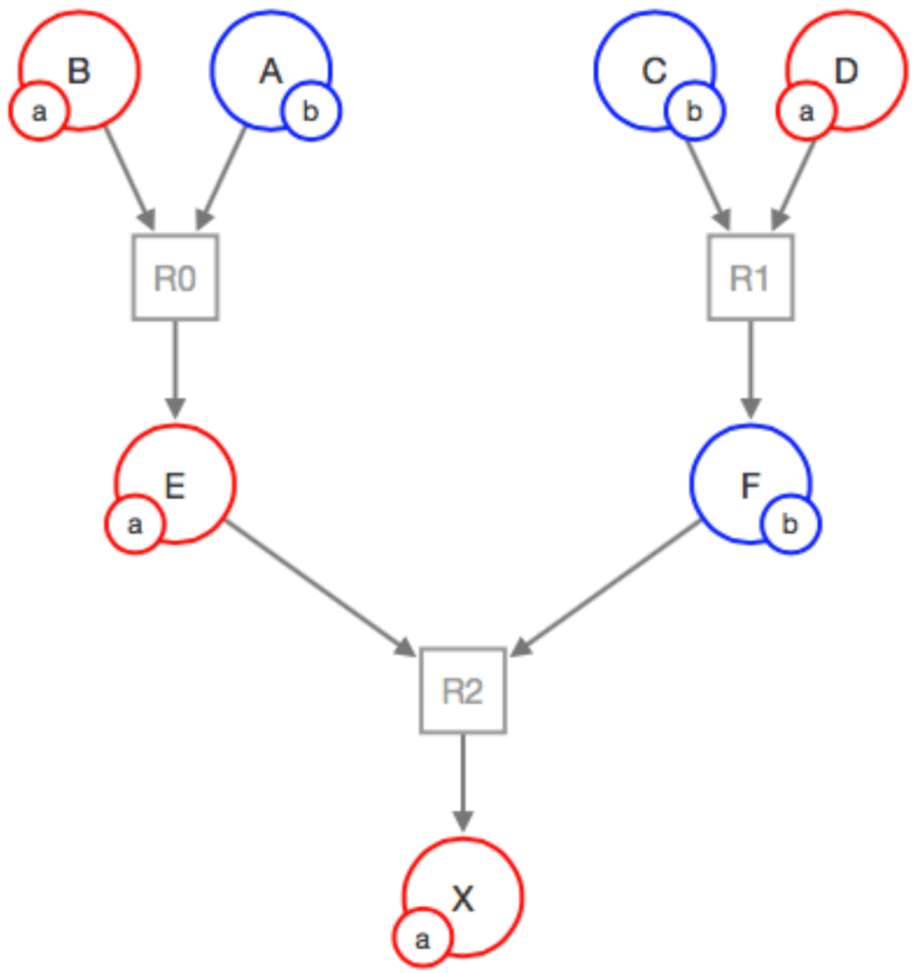
Atomicity

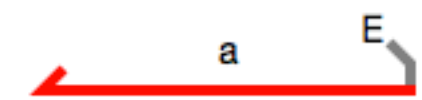




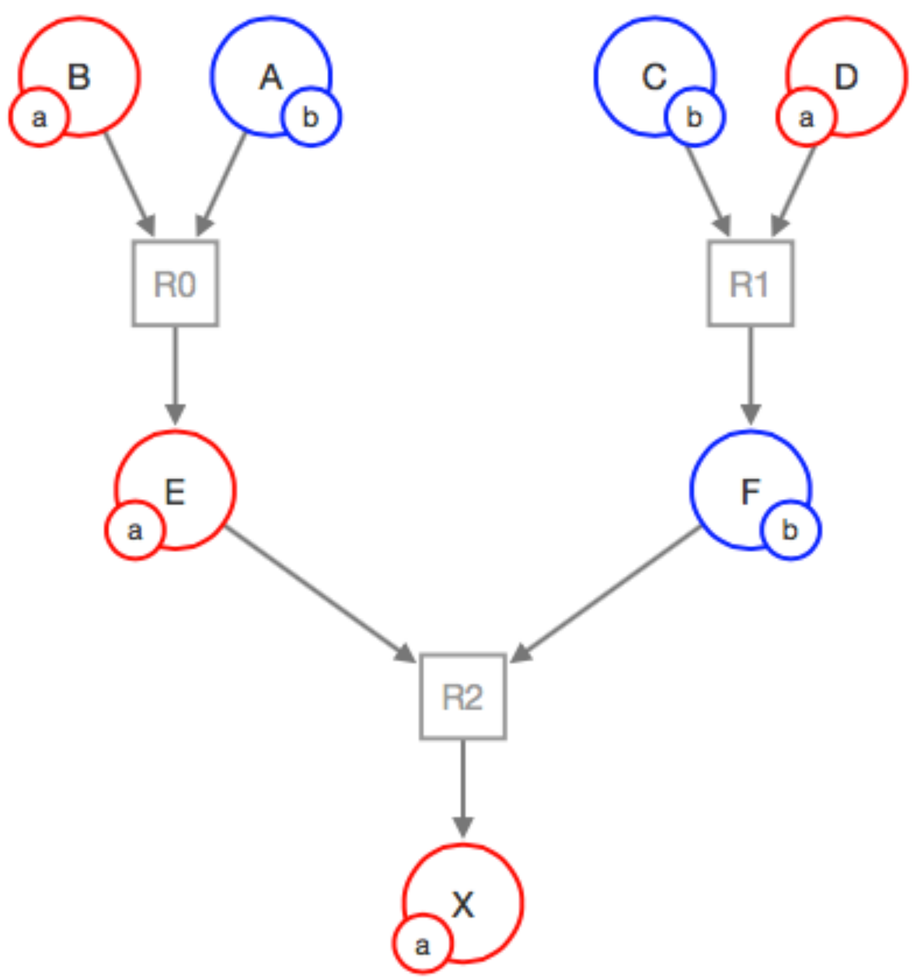


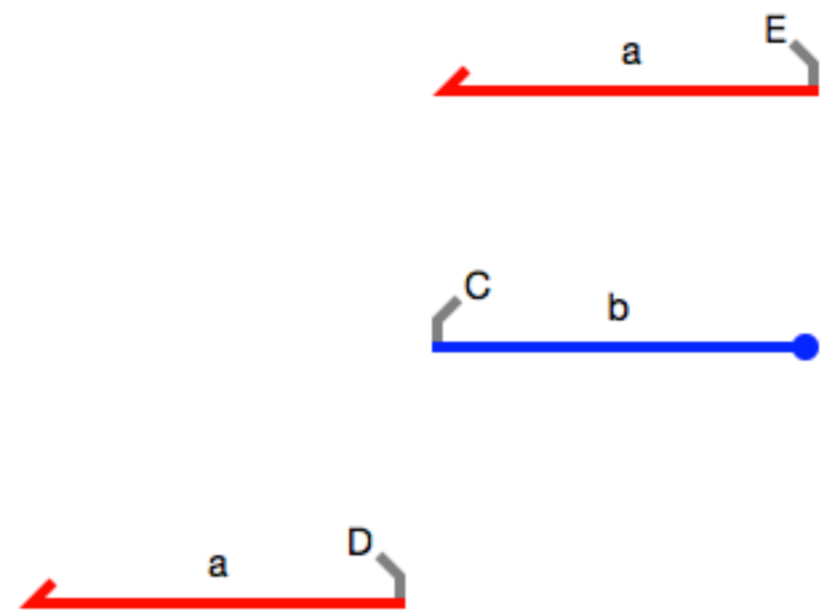
Temporal Protection



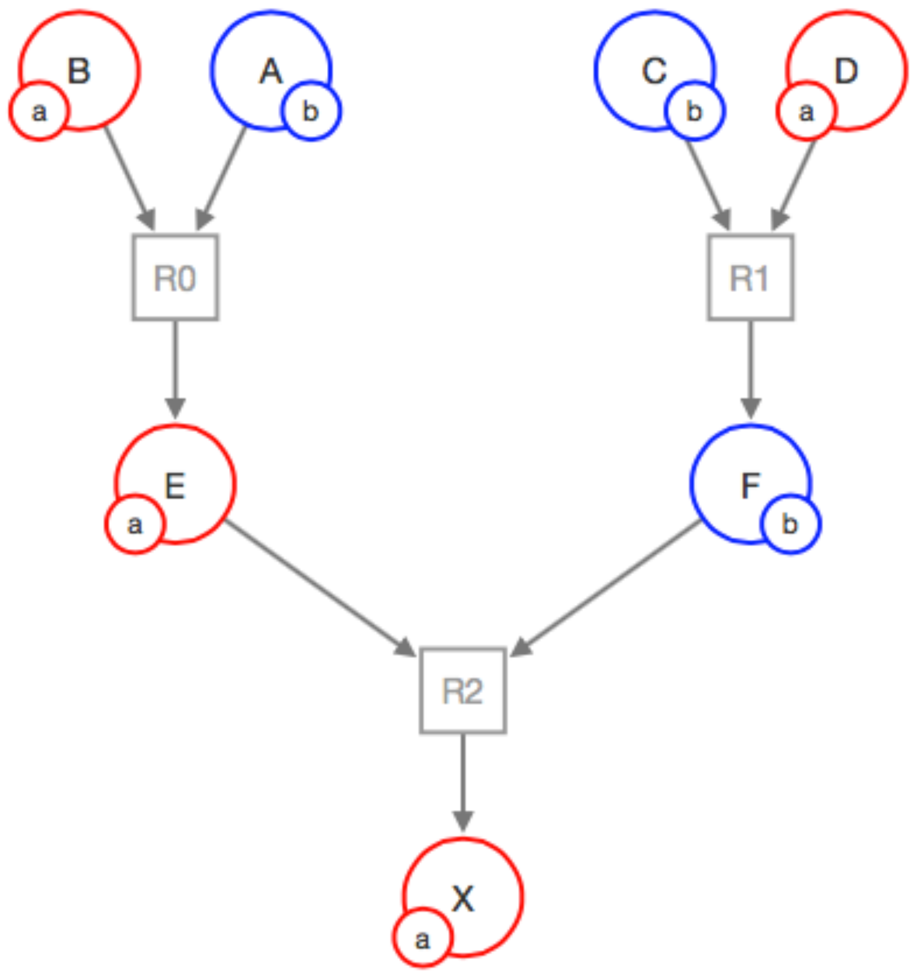


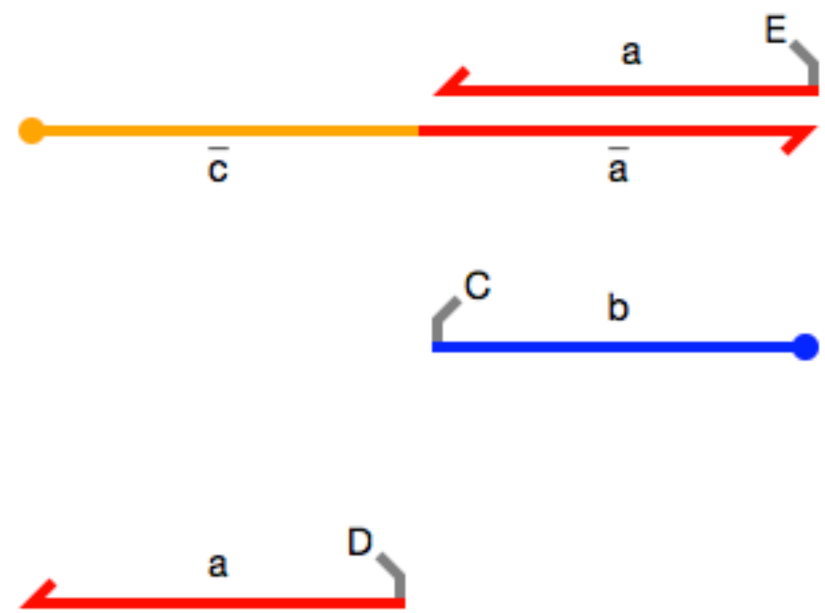
Temporal Protection



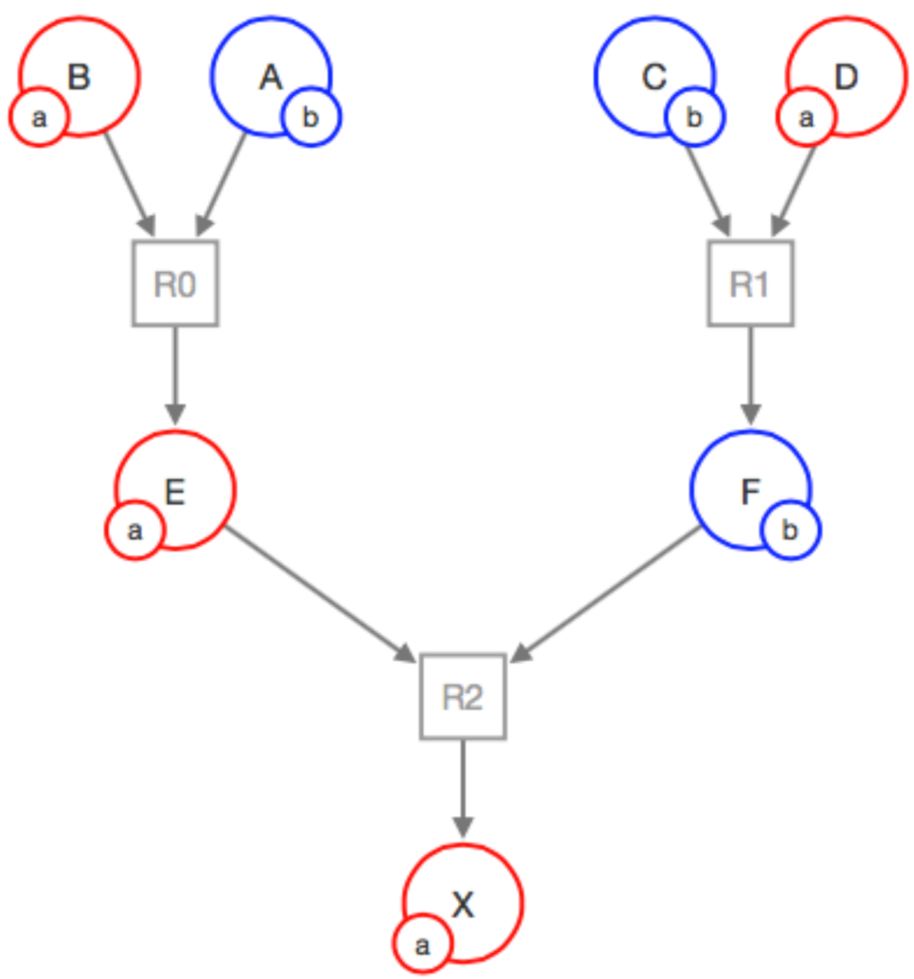


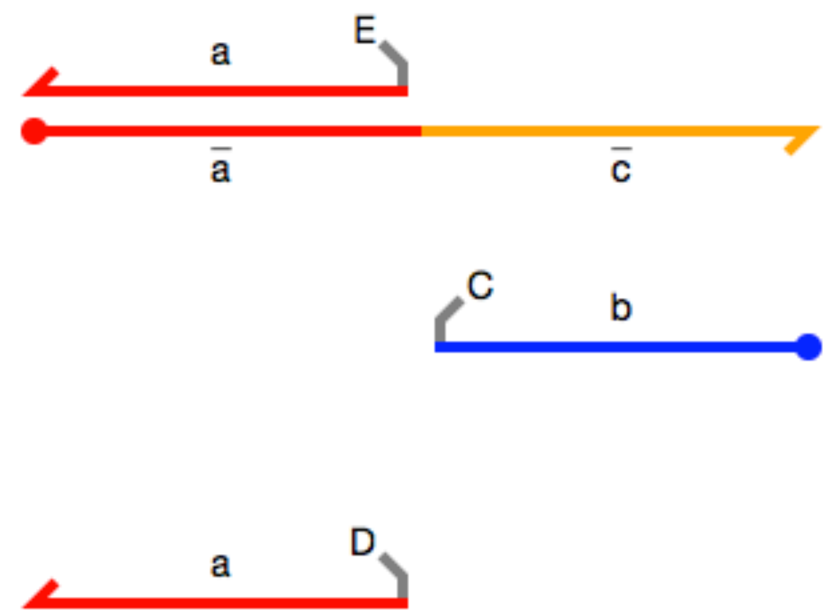
Temporal Protection





Temporal Protection



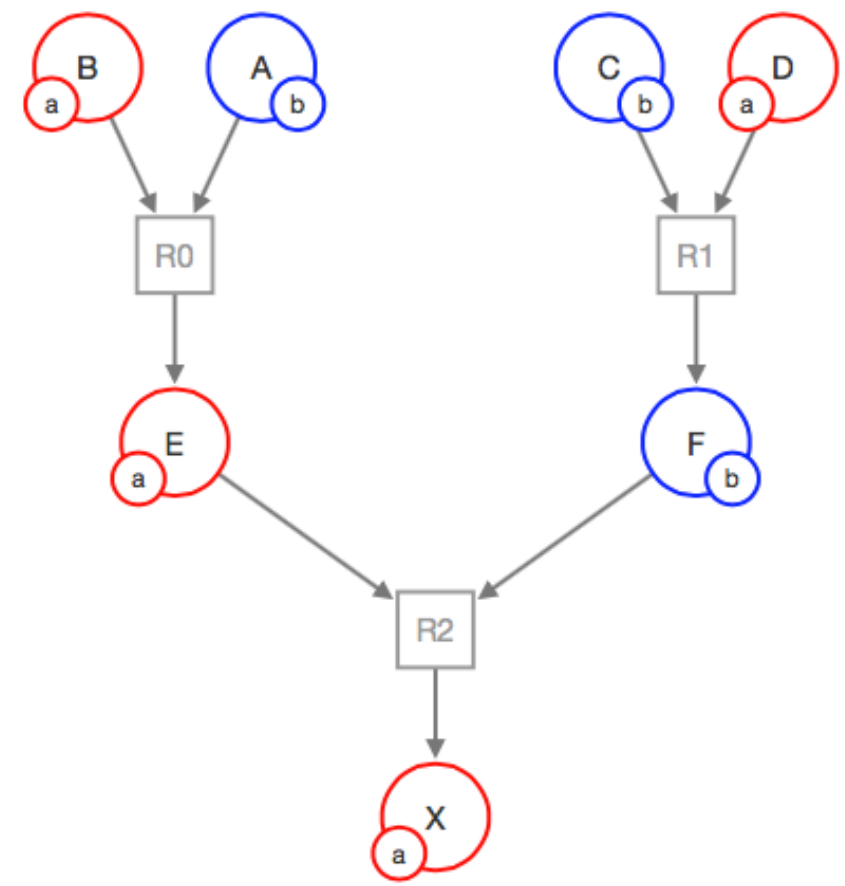


Example

The Algorithm

$$F(node, a, b)$$

$$\min \begin{cases} \max \begin{cases} F(node.left, a, b) \\ F(node.right, a+1, b) \end{cases} \\ \max \begin{cases} F(node.right, a, b) \\ F(node.left, a, b+1) \end{cases} \end{cases}$$



Two Tags on Compounds

- A program with two tags on the compounds and fewest tags overall.
- Runs in polynomial time.

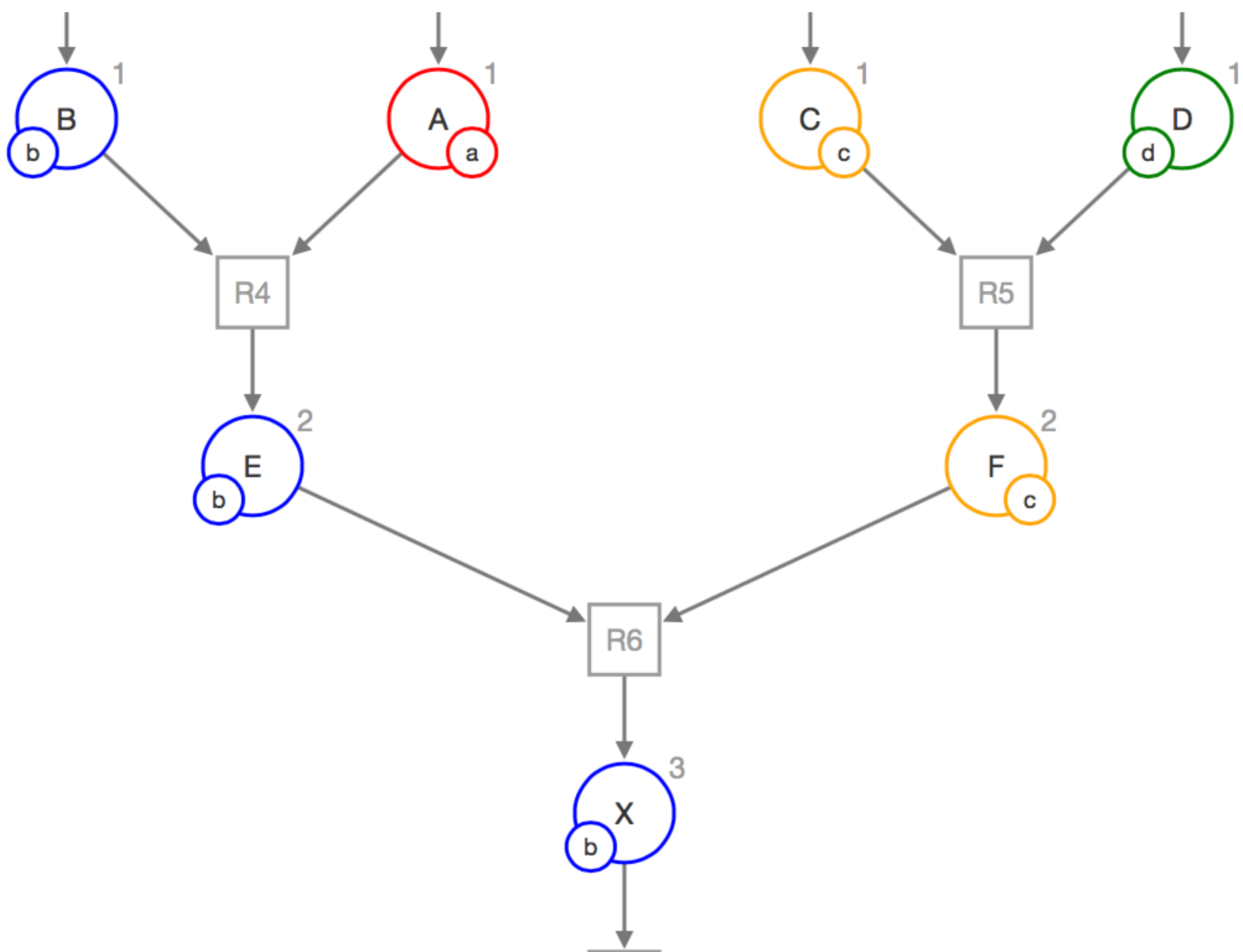
$$O(\underbrace{n}_{\text{nodes}} \cdot \underbrace{h}_{\text{block a}} \cdot \underbrace{h}_{\text{block b}})$$

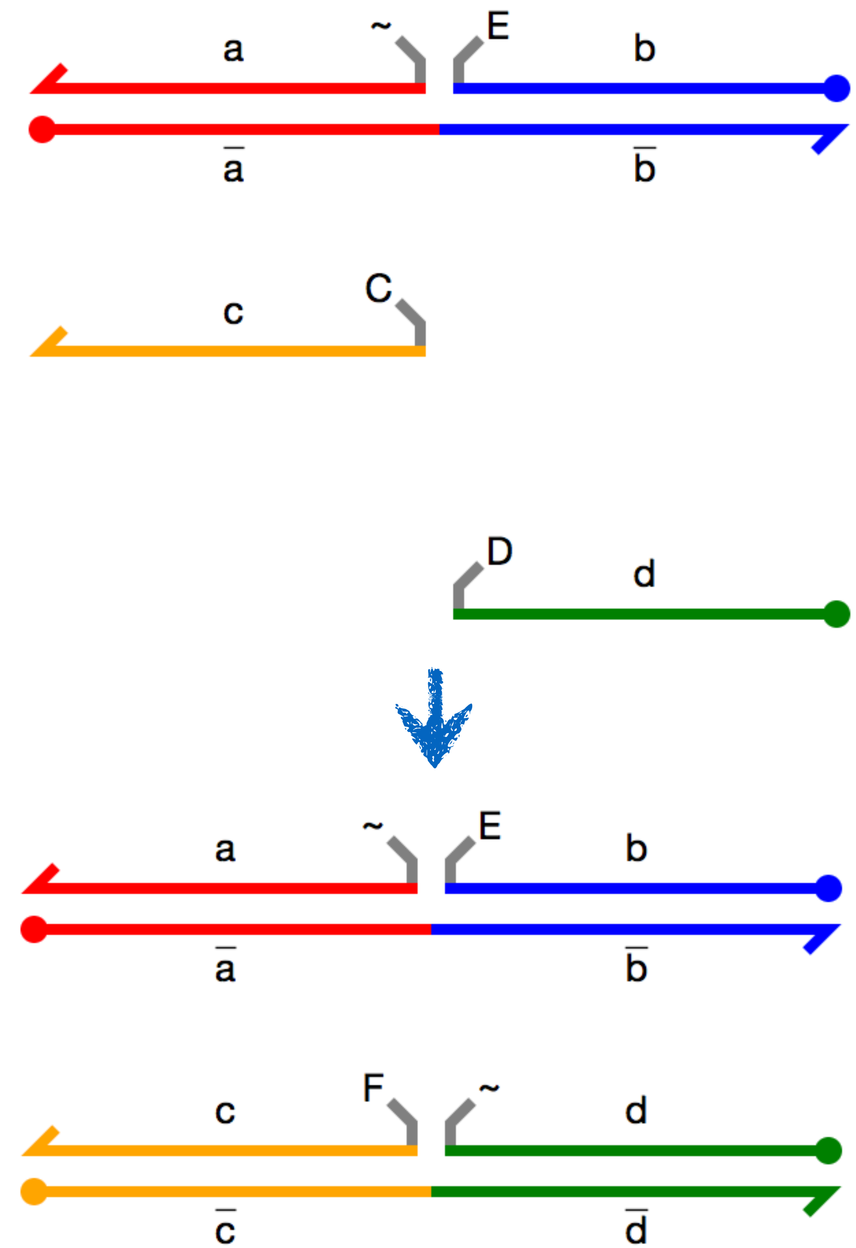
Tags on Compounds	Atomic	Toeholds	Fewest Tags	Shortest Length
2	Yes	No	Polynomial	Linear
2	Yes	Yes	Polynomial	Linear
2	No	No	Not applicable	Not applicable
2	No	Yes	Not applicable	Not applicable
Arbitrary	Yes	No	Polynomial	Linear
Arbitrary	Yes	Yes	*Exponential*	Linear
Arbitrary	No	No	Exponential	Linear
Arbitrary	No	Yes	Exponential	Linear
k	Yes	No	Polynomial	*Exponential*
k	Yes	Yes	*Exponential*	*Exponential*
k	No	No	Exponential	*Exponential*
k	No	Yes	Exponential	*Exponential*

Fewest Tags Without Atomicity

- Atomicity is required for Two Tags on Compounds.
- Let's remove the requirement!

Improved Protection



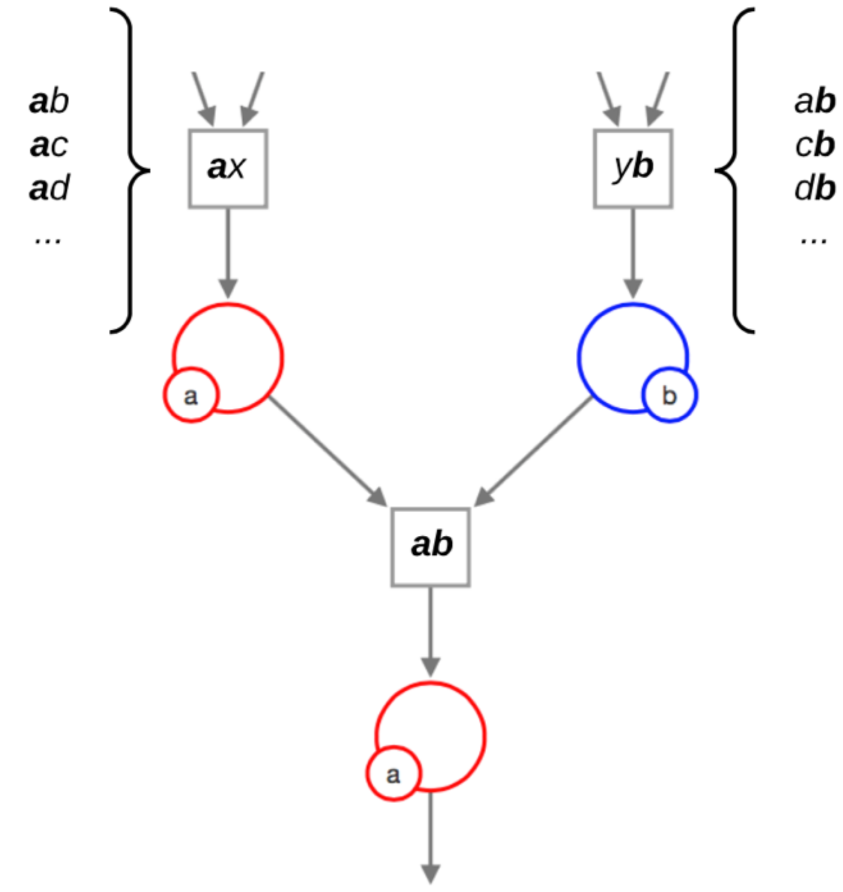


Example

The Algorithm

$F(node, strand, S)$

$$\min \begin{cases} \max \begin{cases} F(node.left, xy, S) & \forall xy \\ F(node.right, zw, S \cup \{xy\}) & \forall zw \end{cases} \\ \max \begin{cases} F(node.right, zw, S) & \forall zw \\ F(node.left, xy, S \cup \{zw\}) & \forall xy \end{cases} \end{cases}$$



The Algorithm

- A program with fewest tags overall.
- Runs in exponential time.

$$O(\underbrace{n}_{\text{nodes}} \cdot \underbrace{\log^2 n}_{\text{strands}} \cdot \underbrace{2^{\log^2 n}}_{\text{blocked}} \cdot \underbrace{\log^2 n}_{\text{inner work}})$$

The Algorithm

- A program with fewest tags overall.
- Runs in exponential time.

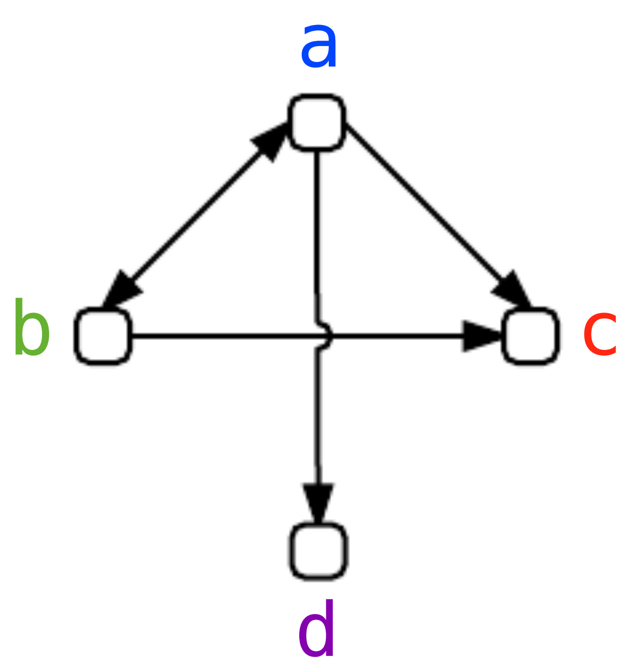
$$O(\underbrace{n}_{\text{nodes}} \cdot \underbrace{\log^2 n}_{\text{strands}} \cdot \underbrace{2^{\log^{\cancel{2}?} n}}_{\text{blocked}} \cdot \underbrace{\log^2 n}_{\text{inner work}})$$

Empirical Analysis

- Assume a program with 4 tags and 5 strands that is solving a synthesis plan.

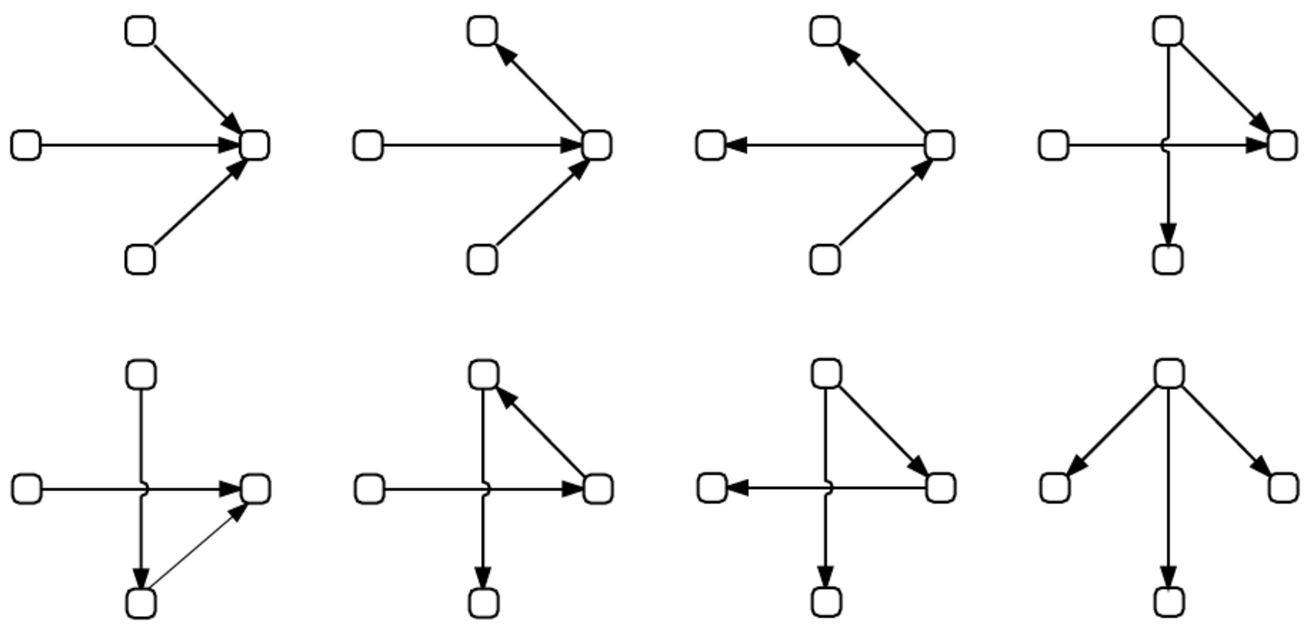
- A solution can be seen as a set of strands.

ab
ba
ac
bc
ad



Strand Sets

- A solution can be seen as the set of strands used.
- Unlabelled connected loop-free digraphs with *#tags* nodes and *#strands* edges.



http://oeis.org/A052283

Strand Sets

- The number of strands that can be made with t tags is

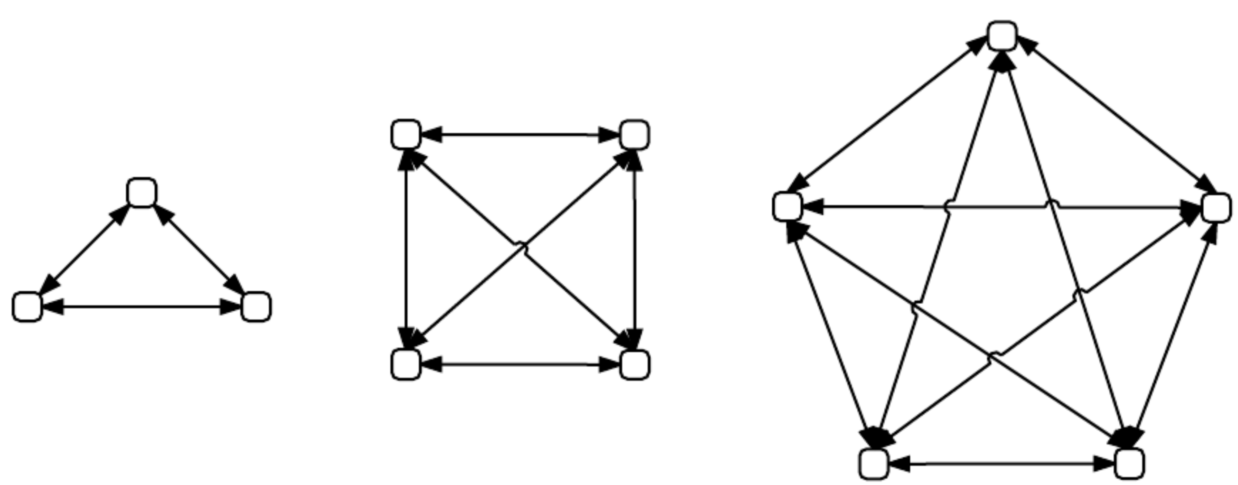
$$|S| = t(t-1)$$

- The number of tags needed to create $|S|$ strands is then

$$t \in \mathrm{O}\left(\sqrt{|S|}\right)$$

Complete Binary Trees

- Can be solved with $\log(n)$ strands.
- The strands are selected consecutively.
- The resulting program has an optimal number of strands and an optimal number of tags.

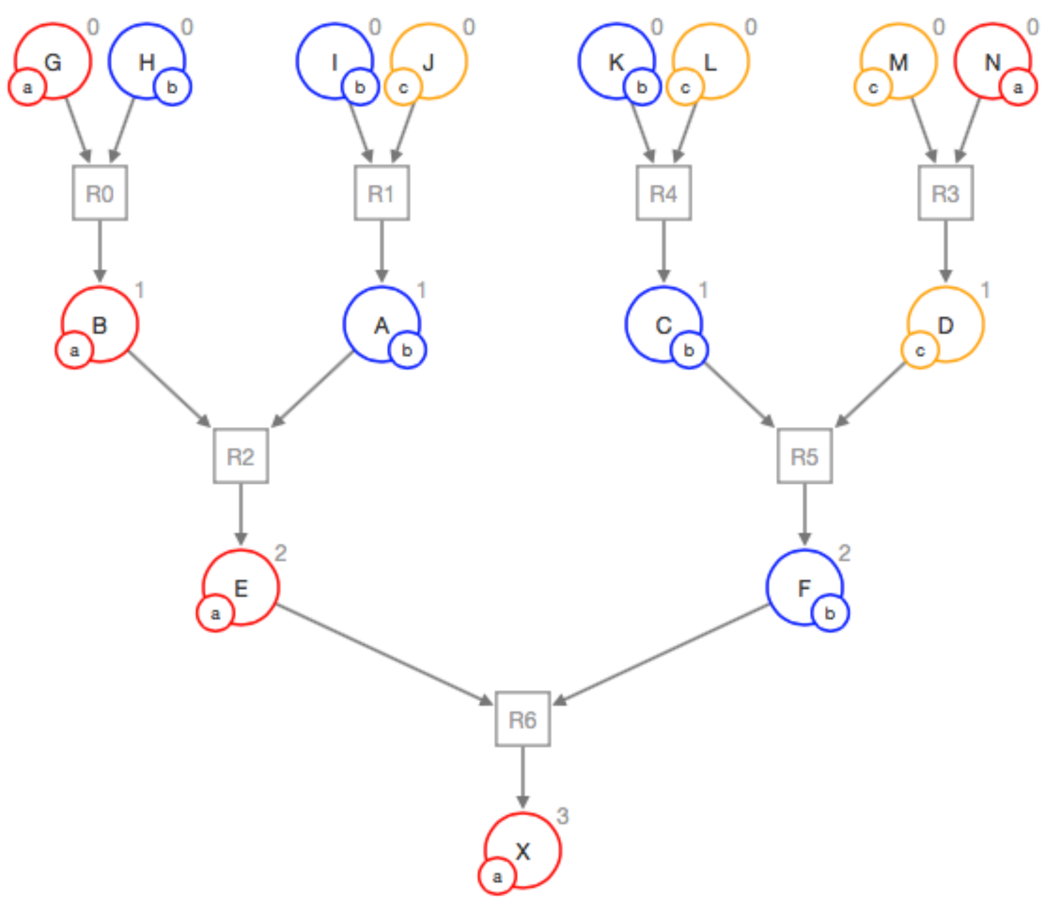


6 12 20

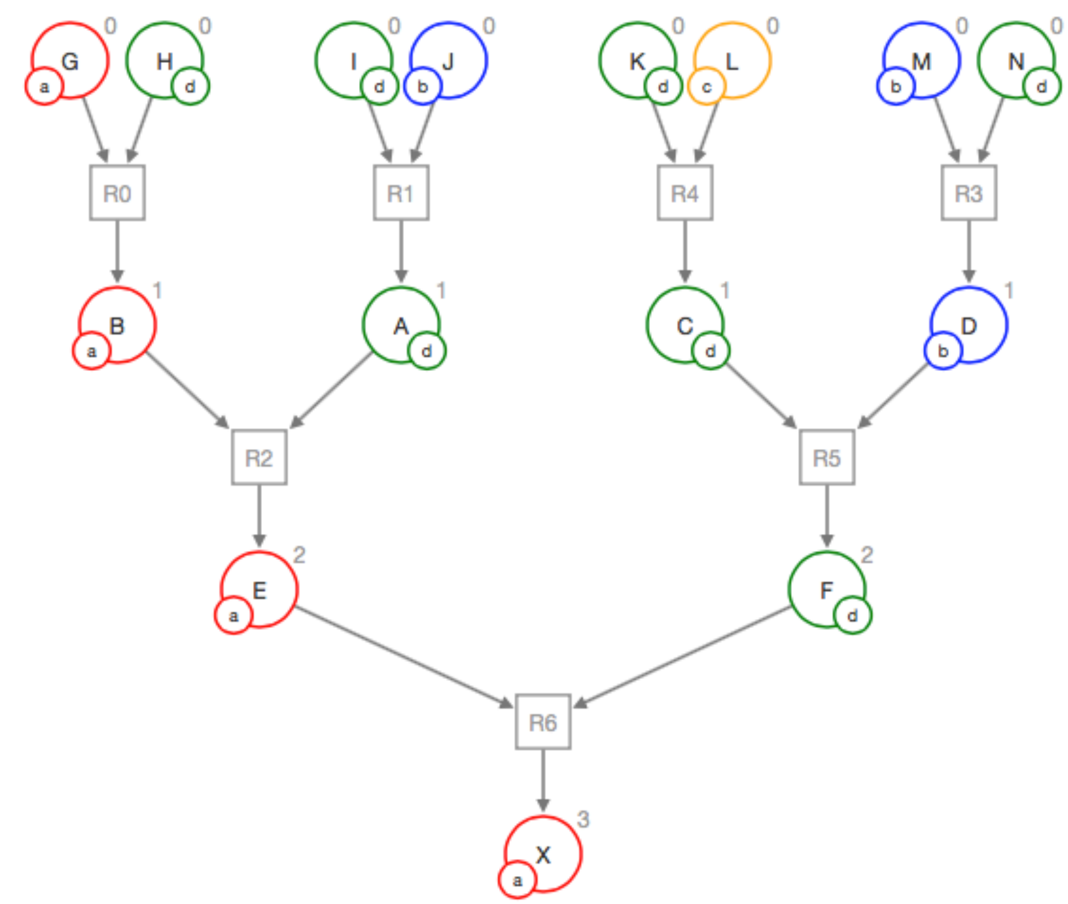
General Binary Trees

- Fix strand set to $S = \{aX, bX, cX, dX, \ldots\}$, $|S| = \log(n)$
- Register allocation in Expression Trees.
- Shows that upper bound is $\log(n) + 1$ tags.
- Approach optimal with regard to no. of strands, but not necessarily with regard to no. of tags.

Comparison



4 strands
3 tags



3 strands
4 tags

Going Polynomial

- Brute force approach.

```
for t = √(log n) ... log n + 1 do
    for |S| = log n ... t(t − 1) do
        sets = select every combination from all (t(t−1) choose |S|) possible
        for S ∈ sets do
            try to find solution using S
        end for
    end for
end for
```

$$
\begin{array}{l}
\textbf{for } t = \sqrt{\log n} \ldots \log n + 1 \textbf{ do} \\
\quad \textbf{for } |S| = \log n \ldots t(t-1) \textbf{ do} \\
\quad\quad sets = \text{select every combination from all } \binom{t(t-1)}{|S|} \text{ possible} \\
\quad\quad \textbf{for } S \in sets \textbf{ do} \\
\quad\quad\quad \text{try to find solution using } S \\
\quad\quad \textbf{end for} \\
\quad \textbf{end for} \\
\textbf{end for}
\end{array}
$$

Going Polynomial

- Optimistic brute force approach, based on empirical analysis.

for $t = \sqrt{\log n} \ldots \sqrt{\log n} + 1$ **do**
    for $|S| = \log n \ldots t(t-1)$ **do**
        $sets$ = select every combination from all $\binom{t(t-1)}{|S|}$ possible
        for $S \in sets$ **do**
            try to find solution using S
        end for
    end for
end for

Going Polynomial

- Optimistic brute force approach, based on empirical analysis.

$\mathrm{O}\left(1\right)$ $\qquad\qquad$ $\mathrm{O}\left(\left(\log n\right)^{\sqrt{\log n}}\right)$

for $t = \sqrt{\log n} \ldots \sqrt{\log n} + 1$ **do**
    for $|S| = \log n \ldots t(t-1)$ **do**
        $sets$ = select every combination from all $\binom{t(t-1)}{|S|}$ possible
        for $S \in sets$ **do**
            try to find solution using S
        end for
    end for
end for

Conclusion

- An ILP formulation for determining tags for programs.
- Known to be polynomial time:
  - Programs of optimal length.
  - Programs with two tags on compounds.
  - Programs with fewest tags (for complete binary trees).
  - Programs with fewest strands.
- "Empirically polynomial":
  - Programs with fewest tags.