Fraglets: Chemical Programming with a Packet Prefix Language

Thomas Meyer and Christian Tschudin
Computer Science Department, University of Basel, Switzerland

15. Kolloquium Programmiersprachen und Grundlagen der Programmierung (KPS'09), Maria Taferl, Oct 13^{th}, 2009

Overview

1. **Functionality** of Fraglets
   - Intro game: Random draws, yet predictable outcome
   - Prefix programs
   - Introductory examples:
     rewriting, FSM, “shuttle service”, active networking
   - A duplicating Quine
2. **Dynamics** of Fraglets
   - See follow up talk by Thomas Meyer

A “Mate-And-Spread” Game

Given: vector of booleans, not uniform `0010011001100`

Do rounds, repeat as long as you wish:

Pick two random positions; If content differs then copy randomly

Question: How will the array look, on average, after some rounds?

A “Mate-And-Spread” Game

Given: vector of booleans, not uniform 0010011001100

Do rounds, repeat as long as you wish:

Pick two random positions; If content differs then copy randomly

```
#define pick(a,b) a=rand()%(len-1);b=rand()%len;if(a==b)a++
main() {char v[]="00100000"; int a, b, len=strlen(v), n=100;
  for (srand(times(0)); n>0; n--) {
    pick(a,b); if (v[a]!=v[b]) {pick(a,b); v[a]='0'; v[b]='1';}
    printf("%s\n", v);
  }
}
```

Question: How will the array look, on average, after some rounds?

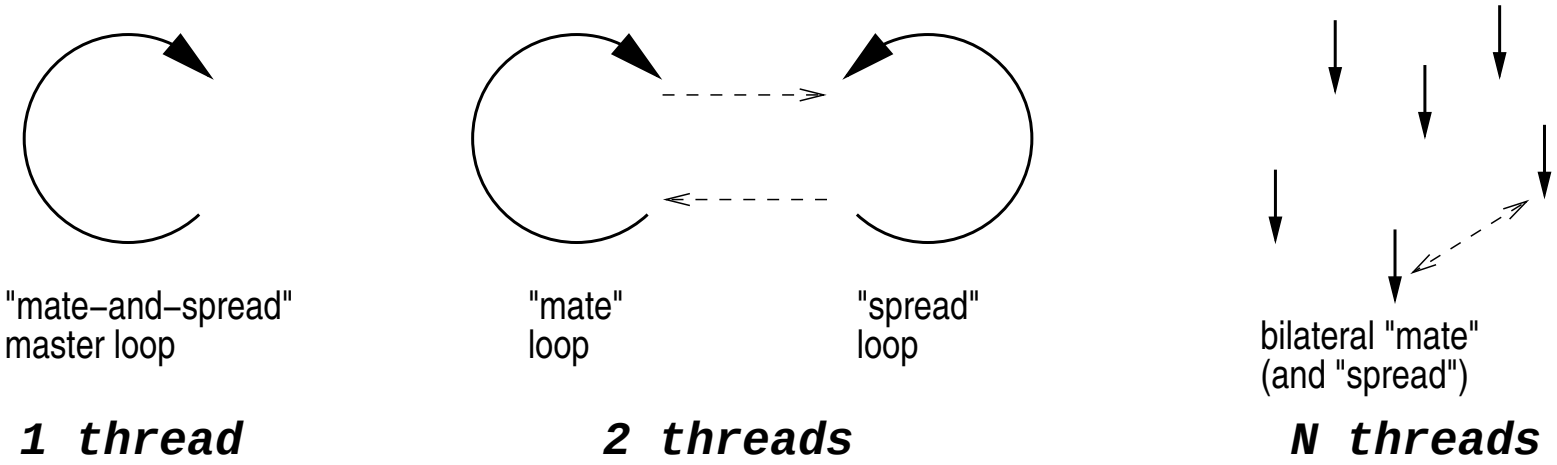
A few “Mate-And-Spread” Games, Results of 100 Rounds

00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000
00100000	10100000	00100000	00100000	00100000	00100000	00100000	00100000
00100000	10100000	00100000	00000100	00100000	00000100	00100000	00000100
00100000	10100000	00100000	00000100	00100000	00000100	00100000	00000100
00100000	10100000	00100000	00000100	00100000	00000100	00100000	00000100
...	...	...	...	...	...	...	...
10010101	11001010	10101001	00101011	10101001	00101011	10101001	00101011
00100000	00100000	00100000	00100000	00100000	00100000	...	*11011110*
00100000	00100010	01000000	00100000	01000000	00000010		11011110
10000000	00100010	01000000	00100000	01000000	00000010		11011110
10000000	00100010	10000000	00100001	10000000	00000010		11011100
10000000	01100010	10000000	00100001	10000000	00000010		11011100
...	...	...	...	...	...		...
10111111	01010001	01101101	11010001	01101101	00110111		**10011001**

Asymptotically, all results will have an equal number of zeros and ones!

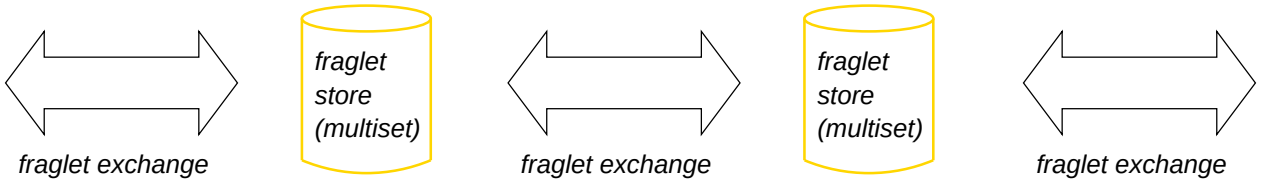
”Mate-and-Spread”: Degrees of Concurrency

- Single thread: sequential program with a master loop
- Two threads (e.g., ”mate” and ”spread”), two loops
- What about `N` threads, and no loops?



Fraglets = *Mobile Computation Fragment*

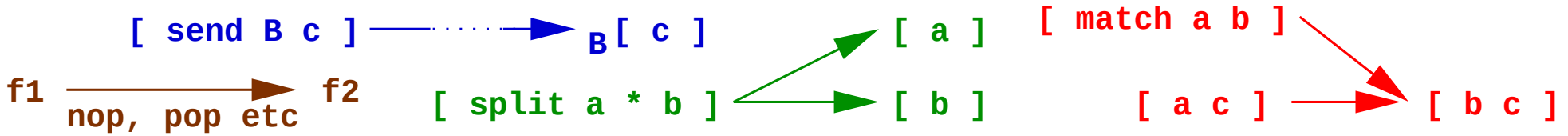
Chemical metaphor: Molecules reacting with each other



- Exchange of fraglets (=molecules, =packets).
  fraglet = word of symbols `[ tag1 tag2 tag3 ... ]`
  fraglet multiset (=reactor, =node)
- Processing rules inside a multiset:
  – reaction (two fraglets merge)
  – transformation (single fraglet rewriting)
  – *processing order (among molecules) is non-deterministic*

Prefix Instructions: a Reaction and Transformations

Op	*input*	*output*
match	`[ match a TAIL1 ], [ a TAIL2 ]`	`[ TAIL1 TAIL2 ]`
nul	`[ nul TAIL ]`	– (fraglet is removed)
nop	`[ nop a TAIL ]`	`[ a TAIL ]`
fork	`[ fork a b TAIL ]`	`[ a TAIL ], [ b TAIL ]`
split	`[ split PART1 * TAIL ]`	`[ PART1 ], [ TAIL ]`
send	$_A$`[ send b TAIL ]`	$_B$`[ TAIL ]` (unreliably)



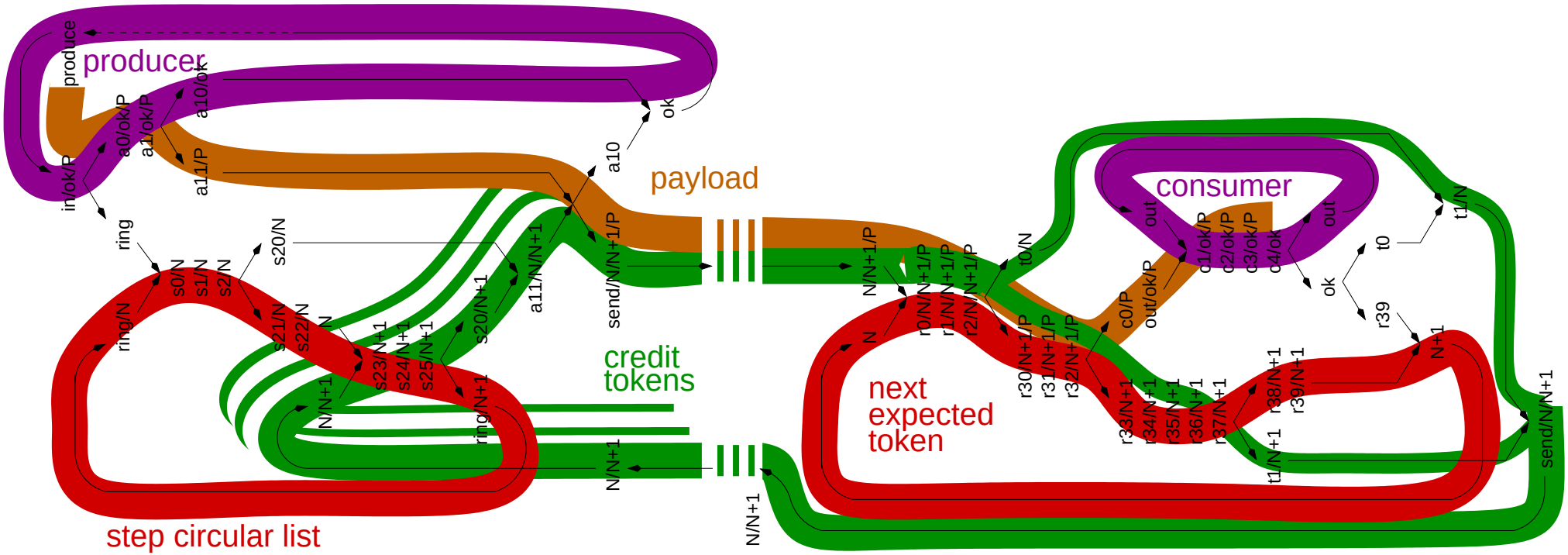
Language has similarities with (is equivalent to) Post Rewriting Systems

Prefix Programs (or Packets)

- Processing rules have a deliberate constraint:
  the first tag fully determines action, no loops
- Origin in networking:
  packet processing = (distributed) header rewriting
- A fraglet's header symbols serve as:
  – instruction
  – related parameters
  – storage
    (the other storage is: other molecules in reactor)

Fraglets Programs are *Distributed Reaction Networks*

Graphical representation of a flow-control protocol



Example 1: Fraglet (header) rewriting

In: [i W]

Out: [o W]

Program: [match i o]

Execution Trace:

$$\left.\begin{matrix}[\ \text{match}\ \ \text{i}\ \ \text{o}\ ] \\ [\ \text{i}\ \ \text{W}\ ]\end{matrix}\right\} \Rightarrow [\ \text{o}\ \ \text{W}\ ]$$

Example 1b: The Temporary `matchp` Hack

Problem: a `match` molecule (the program) is consumed.
Quick hack: Introduce a “persistent” match, as a catalyst.

In: [i W]
Out: [o W]

Program: [match**p** i o]

Execution Trace:

$$\left.\begin{matrix}[\ \text{matchp}\ \ i\ \ o\ ] \\ [\ i\ \ W\ ]\end{matrix}\right\} \Rightarrow [\ o\ \ W\ ], [\ \text{matchp}\ \ i\ \ o\ ]$$

Example 2: (non-deterministic) Finite State Machine

Transitions: [matchp a b]

[matchp b c]

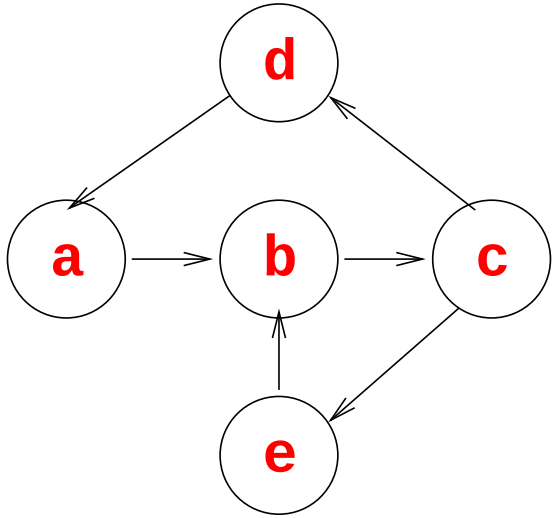
[matchp c d]

[matchp c e]

[matchp d a]

[matchp e b]

State token: [a] *(initial state)*



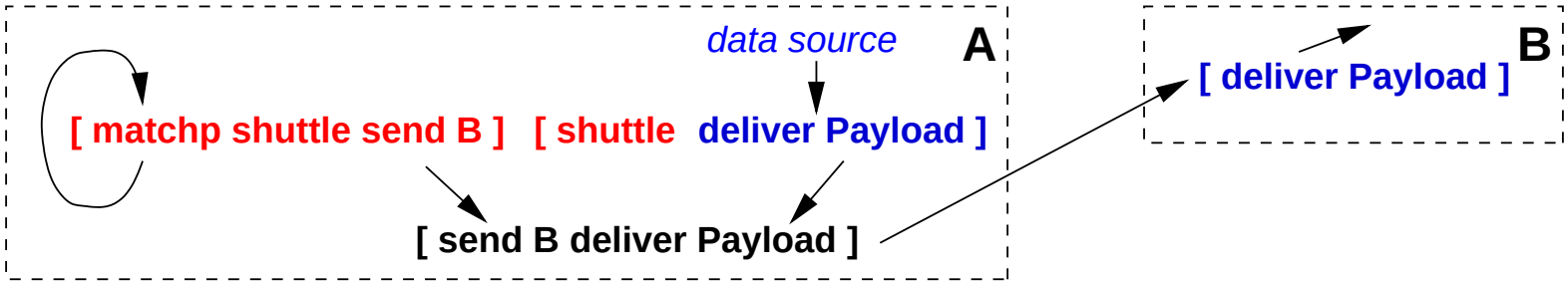
Possible execution trace (visited states):

[a] → [b] → [c] → [e] → [b] → [c] → [d] → [a] ...

Example 3: A “Shuttle Service”

Service req: [**shuttle** deliver Payload]

Shuttle srv: [**matchp shuttle** send B]



Execution Trace:

$_A$[shuttle deliver Payload] $\Rightarrow$ $_A$[send B deliver Payload]

$\Rightarrow$ $_B$[deliver Payload]

Example 4: Active Networking (and Source Routing)

Networking style called **“Active Networking”**:

– packets contain processing logic

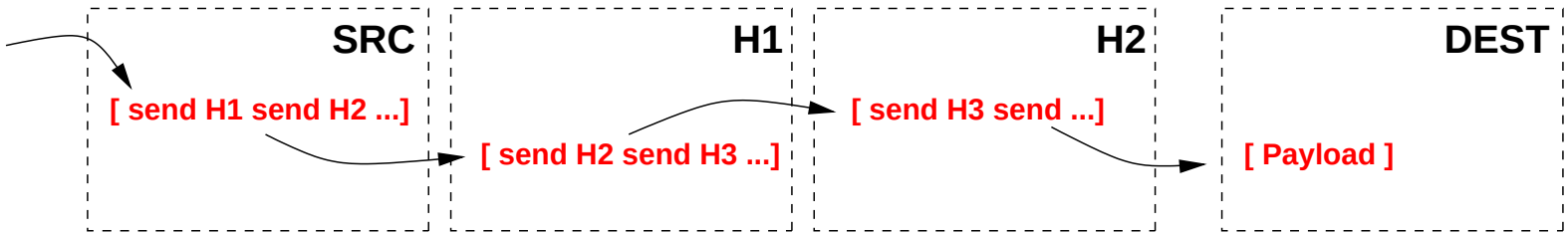
– advantage: minimal preinstalled protocol software requirements

Source routing – the source node prescribes the path

– via H1, H2, ... to some `Dest`

– Simple active networking (inband) version:

```
[ send H1 send H2 send H3 ...  send DEST Payload ]
```



Example 5: Quines

- Classics of informatics:
  – Write a program that outputs its own source code.
  – Named after Willard van Orman Quine (1908–2000)
  – “Quines” exist for every turing complete language
- How to write a Quine for a prefix language?

Basic Quine structure:
– needs a “blueprint”
– active part (is using the blueprint, and is its activated form)

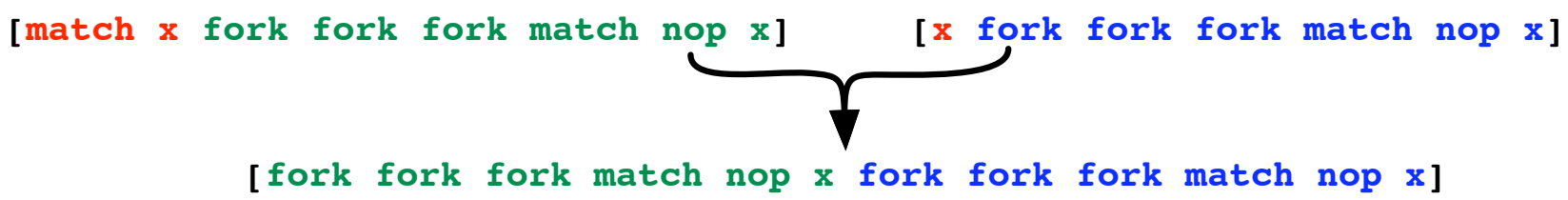
Plus here: Our Quine duplicates (needed in second part of the talk)

A Duplicating Quine for Fraglets (1)

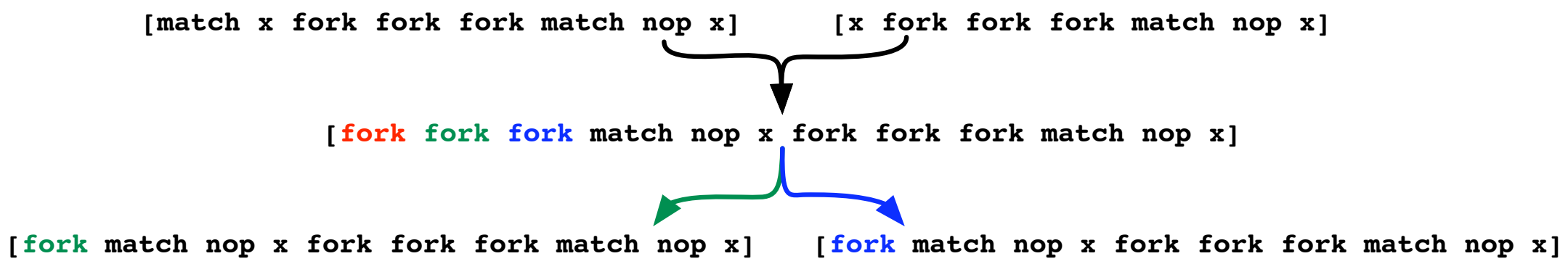
```
[match x fork fork fork match nop x]
```

```
[x fork fork fork match nop x]
```

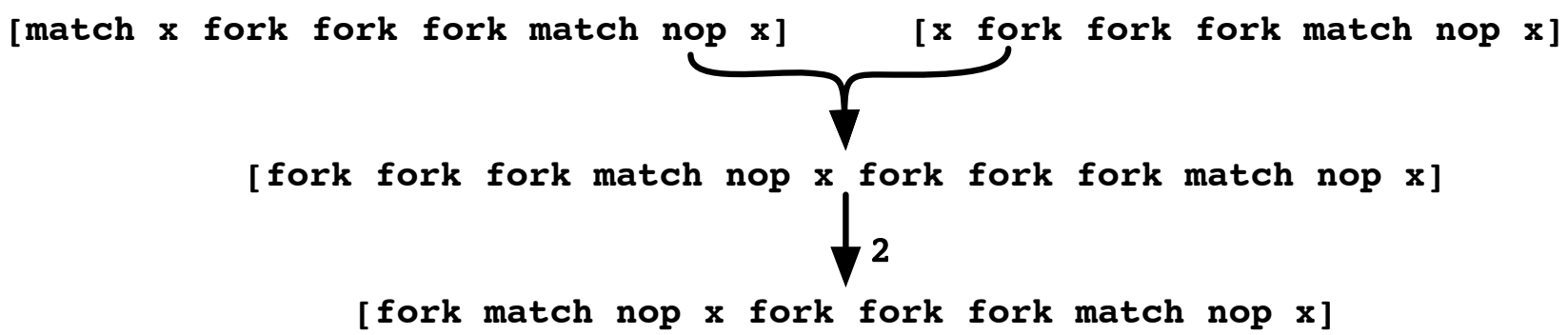
A Duplicating Quine for Fraglets (2)



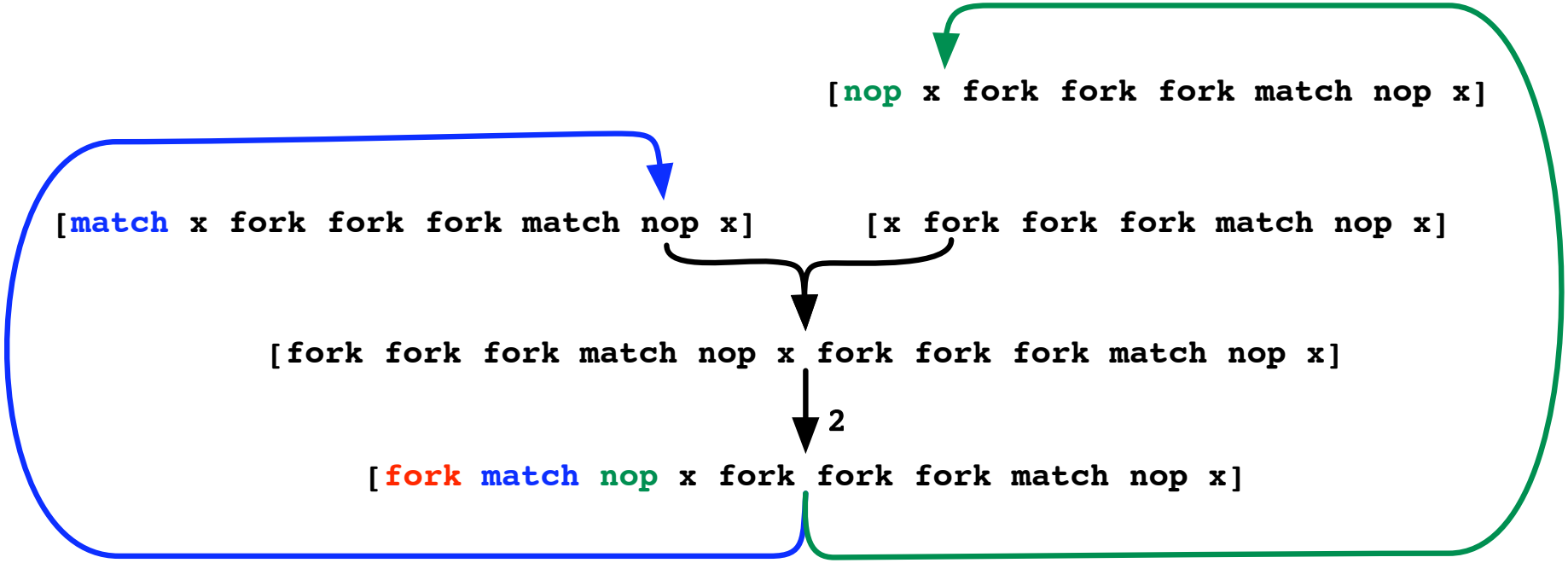
A Duplicating Quine for Fraglets (3a)



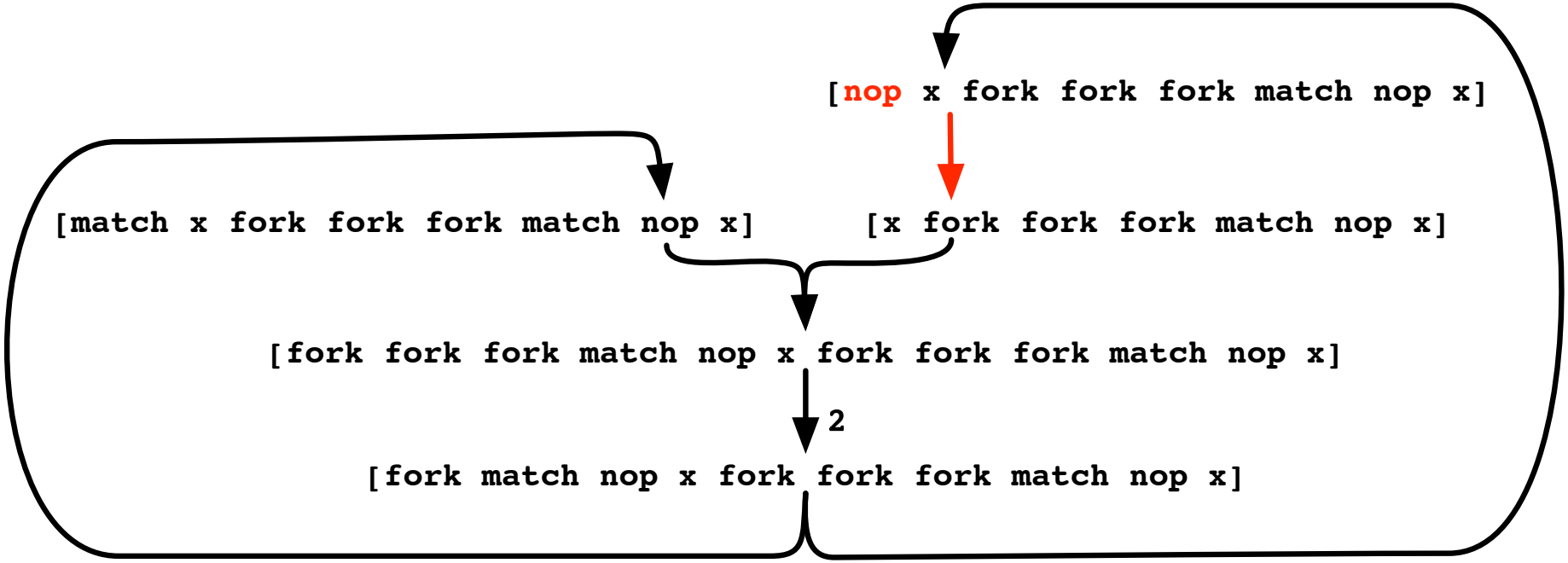
A Duplicating Quine for Fraglets (3b)



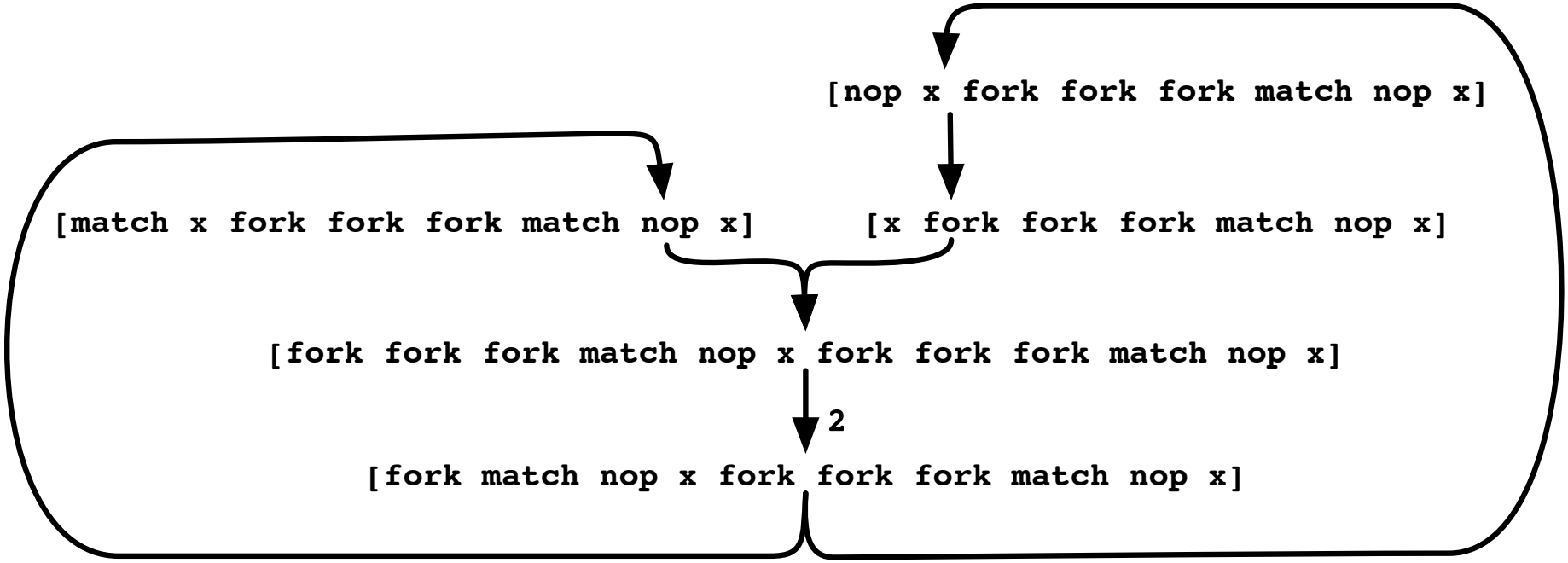
A Duplicating Quine for Fraglets (4)



A Duplicating Quine for Fraglets (5)



A Duplicating Quine for Fraglets (6)



This is an *autocatalytic* reaction network, no need for `matchp`

Conclusions

- Metaphor of “chemical computations”
  – not wet-lab: string rewriting
  – inherently parallel and non-deterministic execution
  We wish to exploit emergent properties of chemical systems like self-organization (see Turing’s work), or robustness
- Active networking, prefix programs: code = data = molecule
- Quines also work that way: what is data and what is code *is mere attributation*, and *not* an intrinsic property.

Next: Dynamics of Fraglets – Remember the mate-and-spread game!