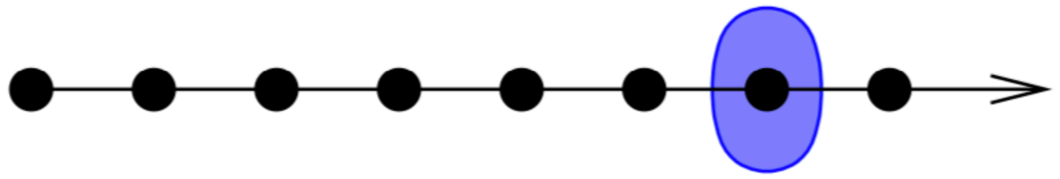
LTL Model Checking in Spin

11.04.2020

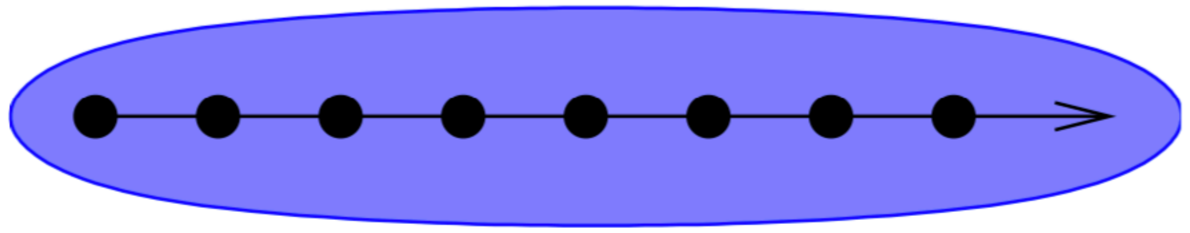
Refresh

finally p



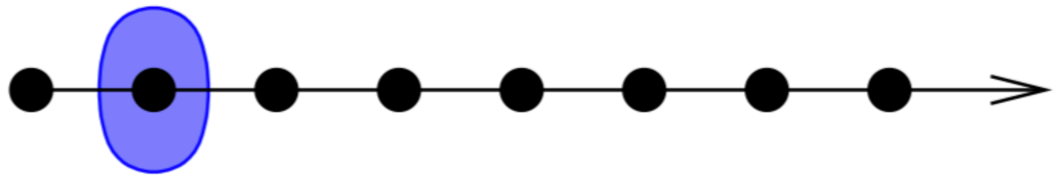
F p

globally p



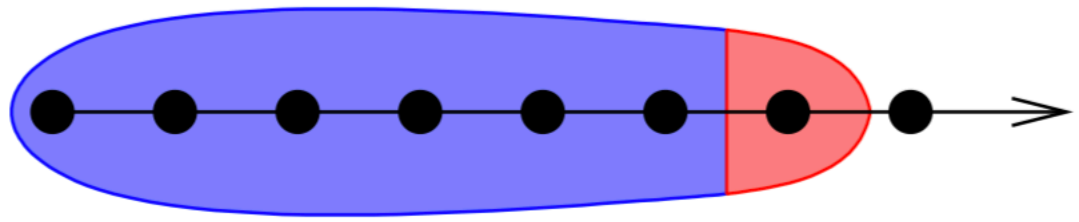
G p

next p



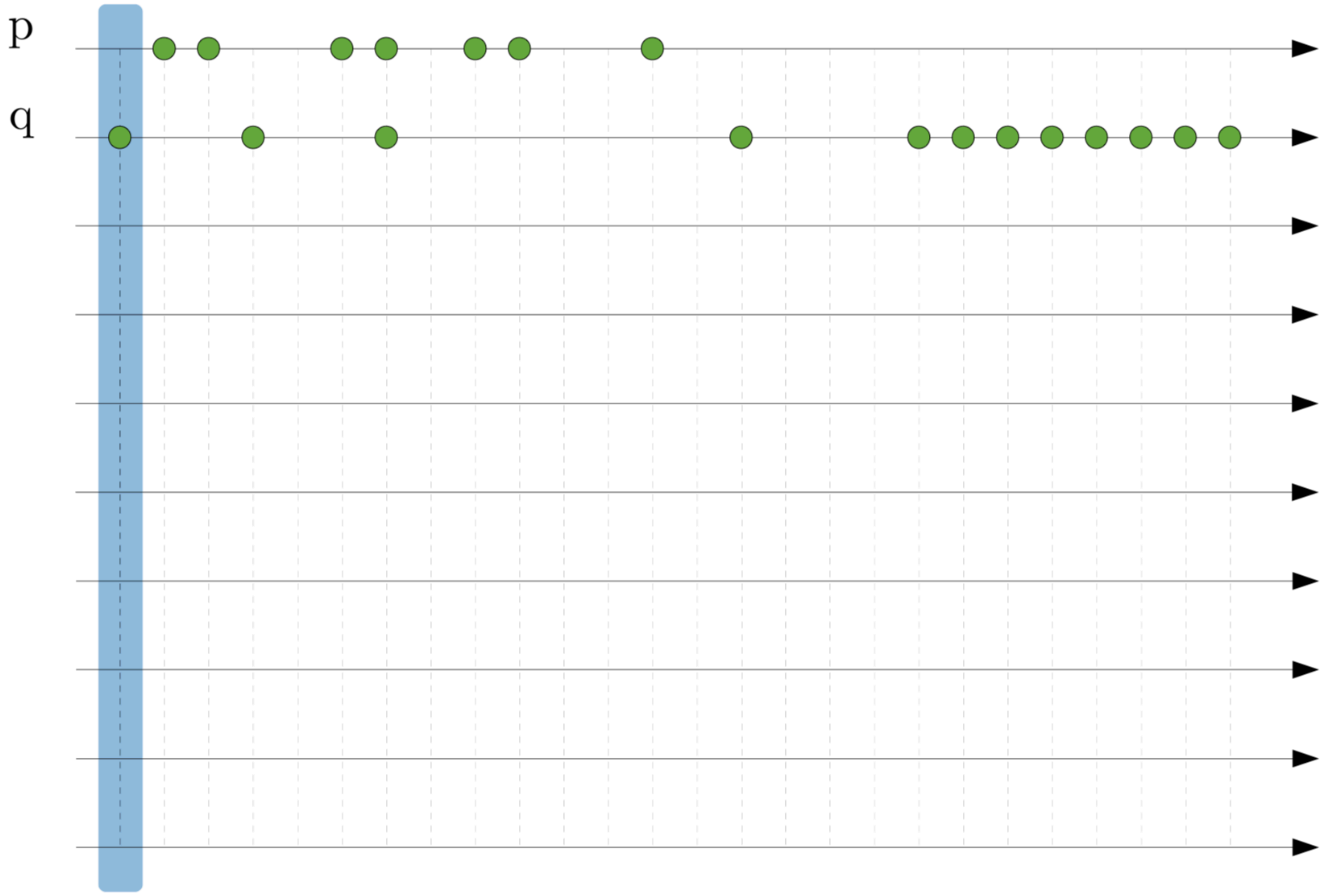
X p

p until q

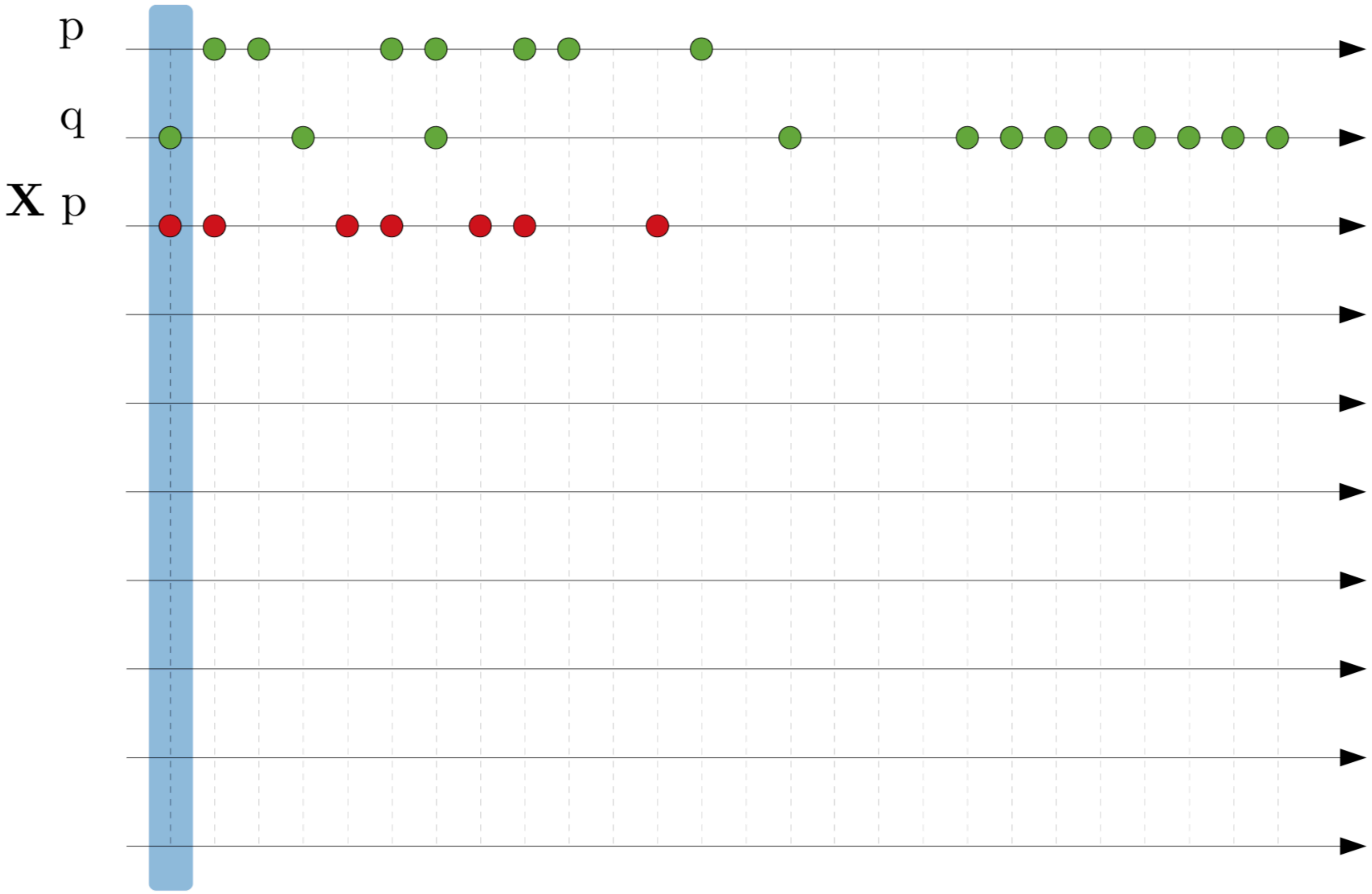


p U q

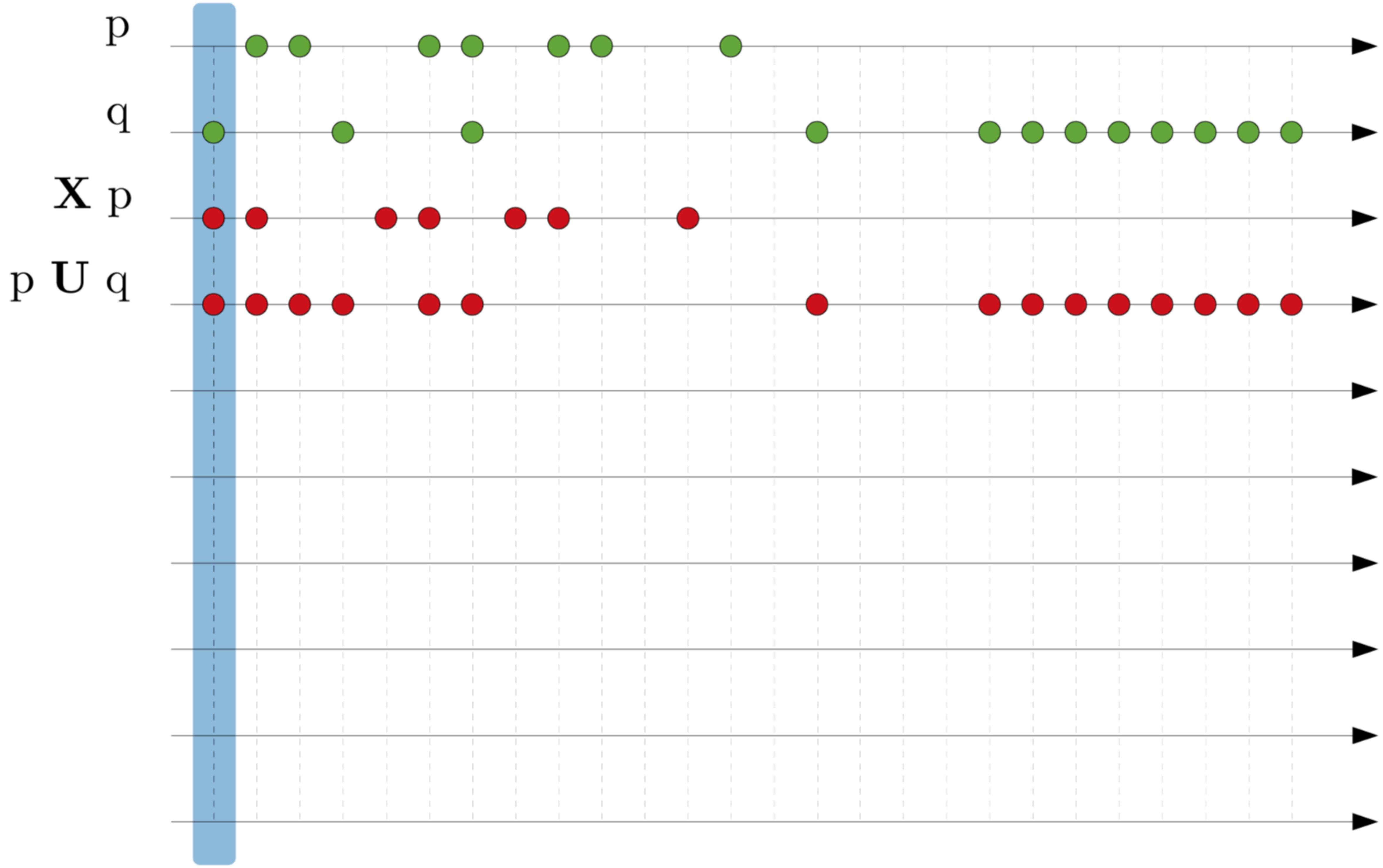
Refresh



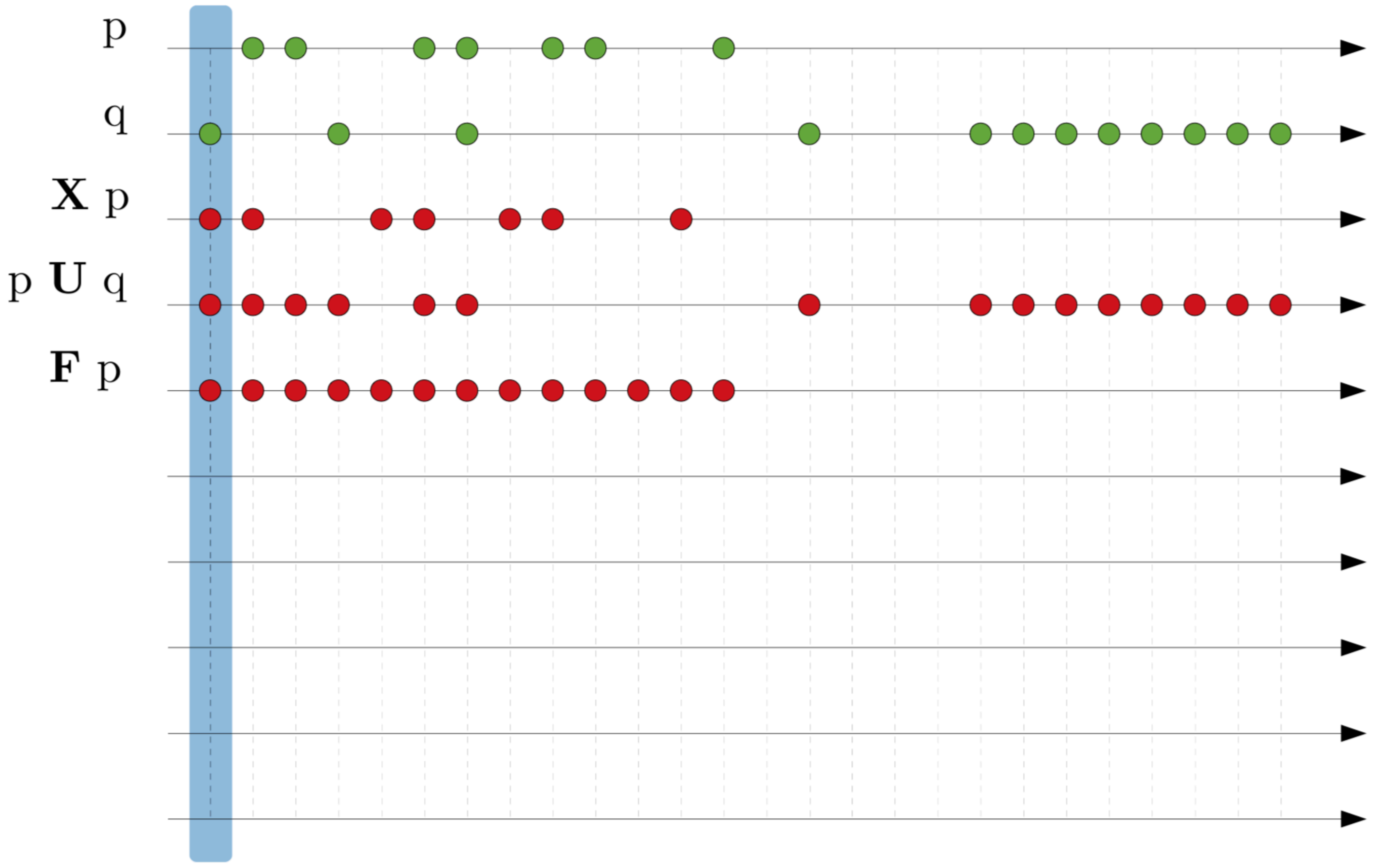
Refresh



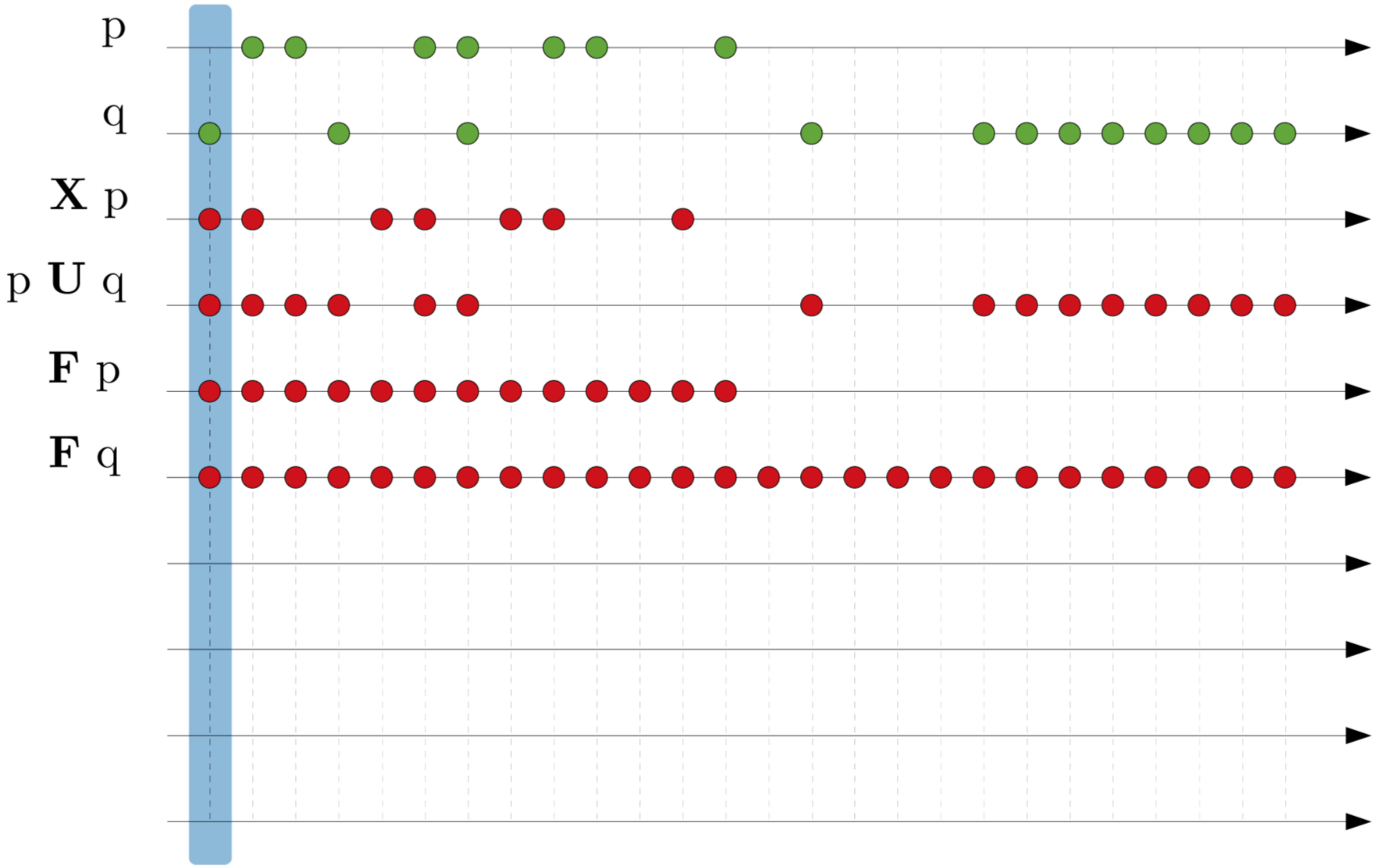
Refresh



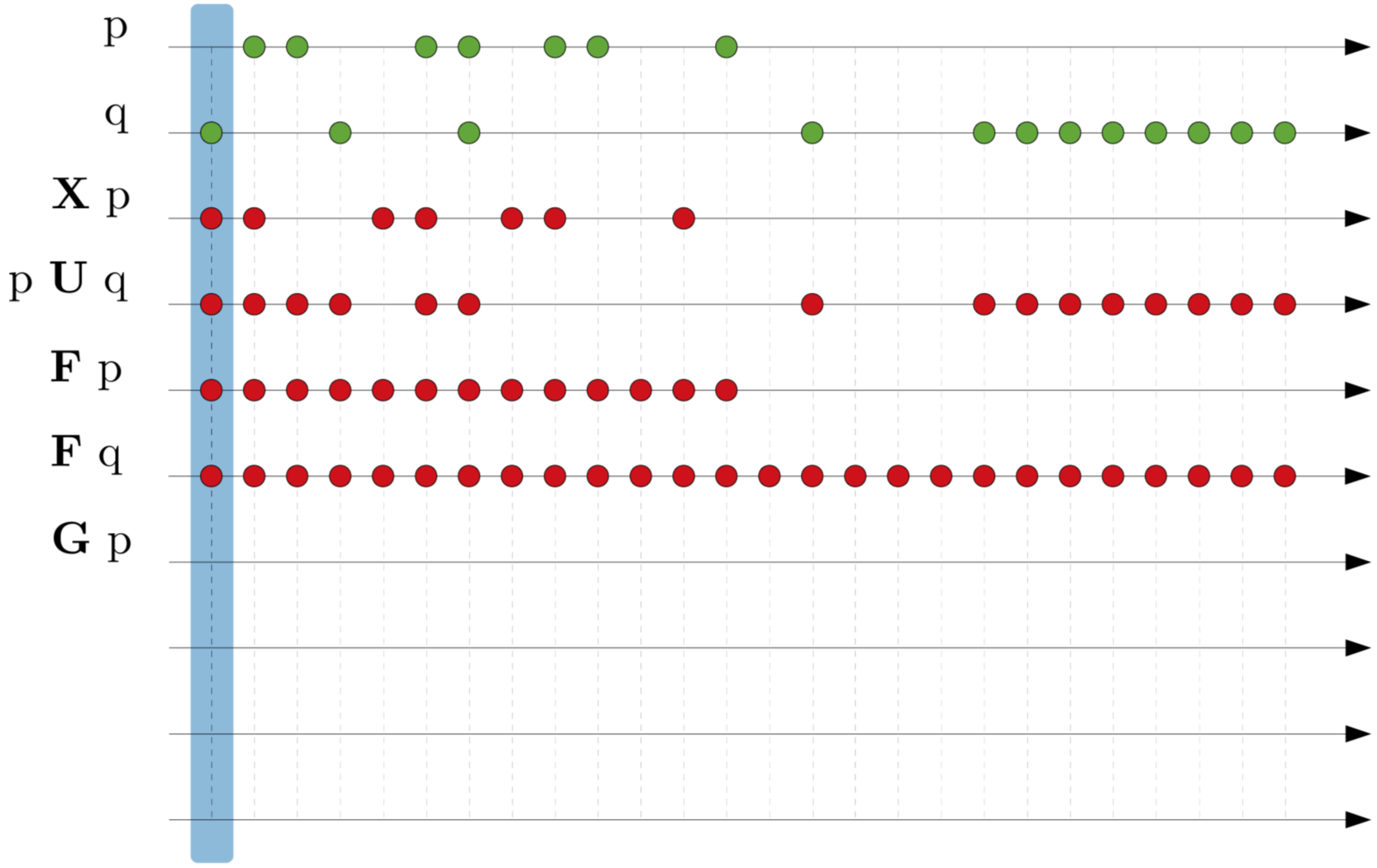
Refresh



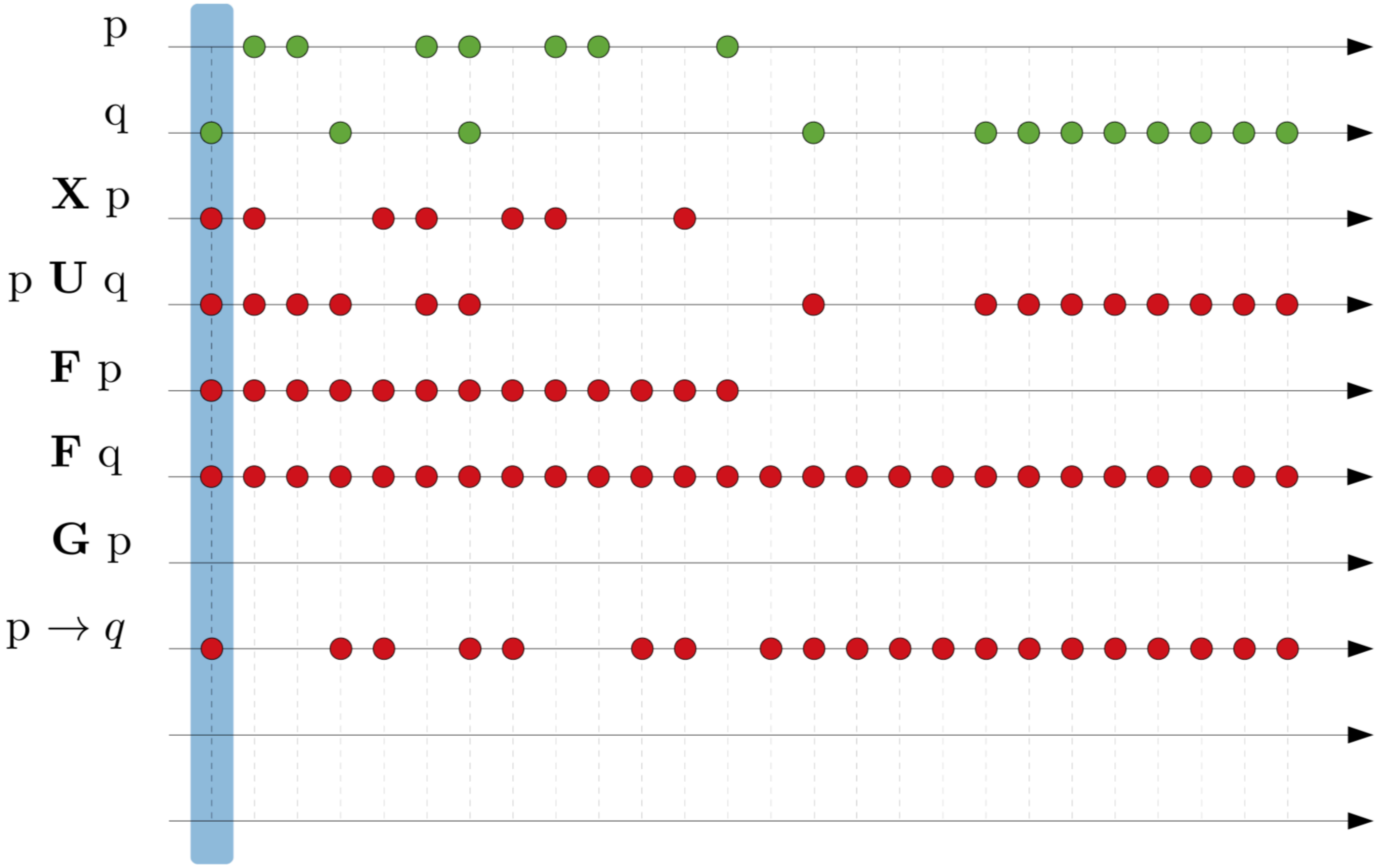
Refresh



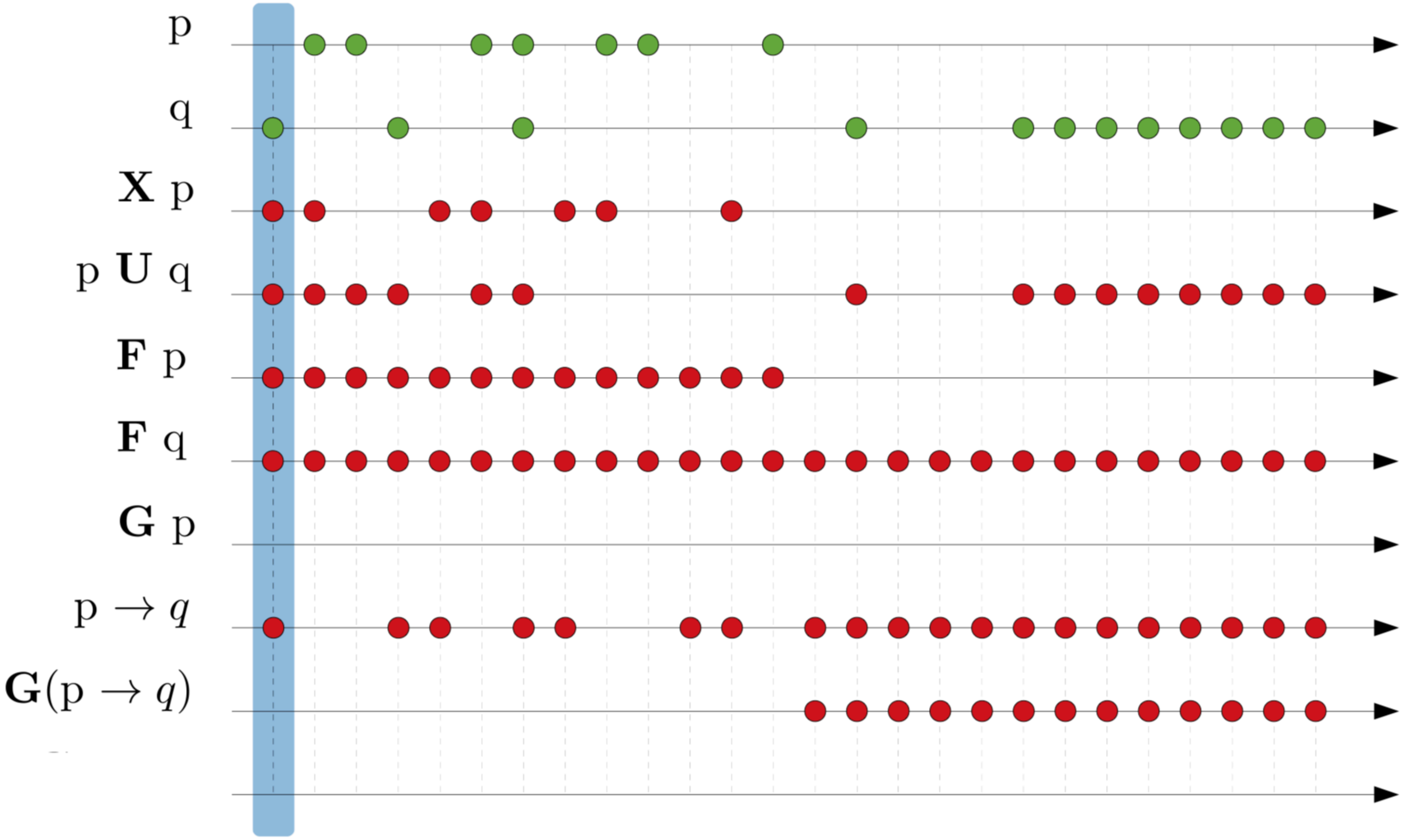
Refresh



Refresh

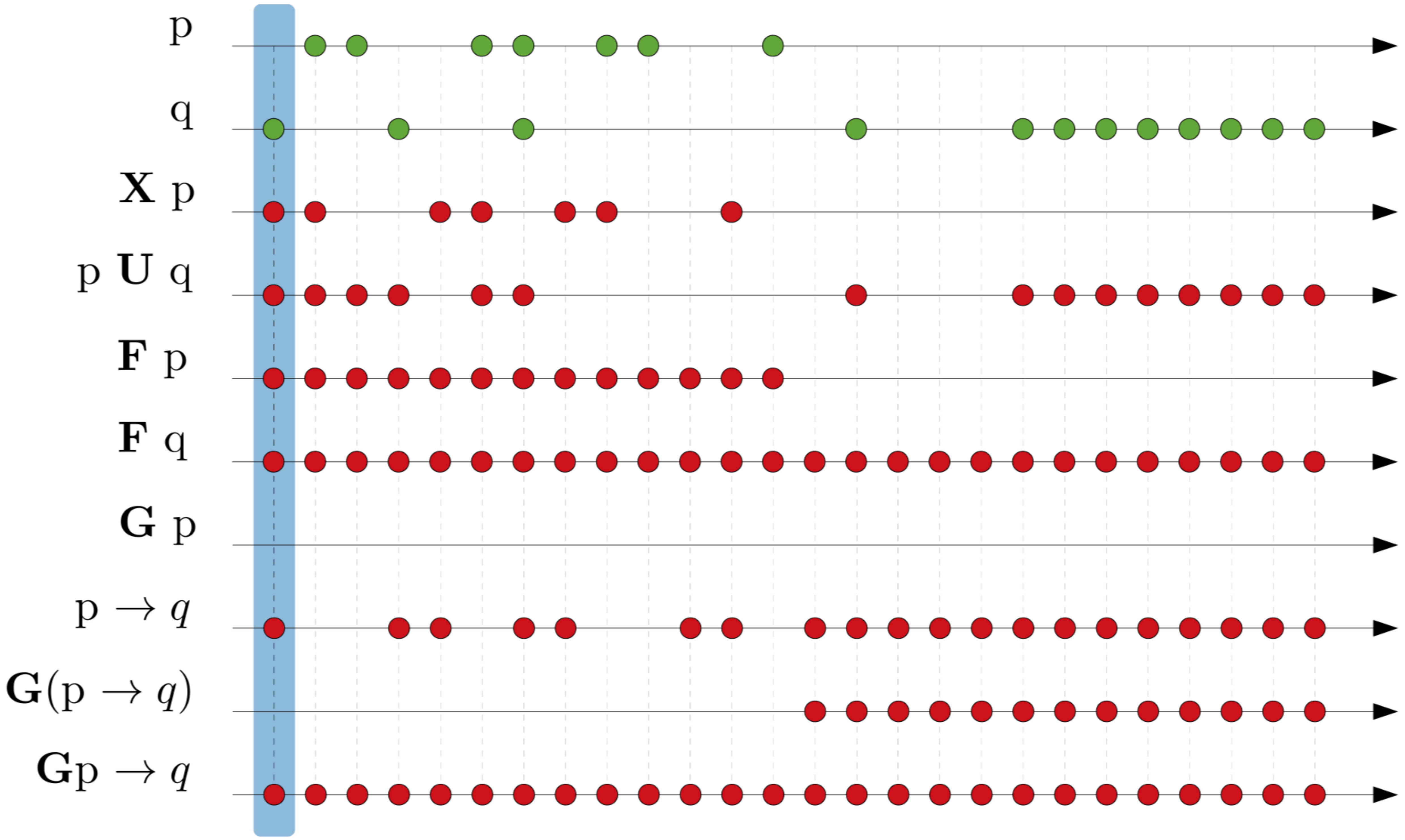


Refresh



p
q
$\mathbf{X}\ p$
$p\ \mathbf{U}\ q$
$\mathbf{F}\ p$
$\mathbf{F}\ q$
$\mathbf{G}\ p$
$p \rightarrow q$
$\mathbf{G}(p \rightarrow q)$

Refresh



p
q
$\mathbf{X}\ \mathrm{p}$
$\mathrm{p}\ \mathbf{U}\ \mathrm{q}$
$\mathbf{F}\ \mathrm{p}$
$\mathbf{F}\ \mathrm{q}$
$\mathbf{G}\ \mathrm{p}$
$\mathrm{p} \rightarrow q$
$\mathbf{G}(\mathrm{p} \rightarrow q)$
$\mathbf{G}\mathrm{p} \rightarrow q$

LTL Syntax in Spin

- **Grammar**:
  - `ltl ::= opd | ( ltl ) | ltl binop ltl | unop ltl`
- `opd`:
  - `true`, `false`, and user-defined names starting with a lower-case letter
- `unop`:
  - `[]`: globally/always
  - `<>`: finally/eventually
  - `!`: not
  - `X`: next
- `binop`:
  - `U`: until
  - `V`: release
  - `&&`: and
  - `||`: or
  - `->`: implication
  - `<->`: equivalence

remember: $(\varphi V \psi) = !(!\varphi U !\psi)$

Short example

Example (foo.pml): verify that `b` is always true.

```
bool b = true;

active proctype main() {
    printf("hello world!\n");
    b = false;
}
```

Short example

Example (foo.pml): verify that `b` is always true.

```
bool b = true;

active proctype main() {
    printf("hello world!\n");
    b = false;
}
```

Standard Approach:

- add the LTL formula in `foo.pml`:

  ```
  ltl p1 { [] b }
  ```

Short example

Example (foo.pml): verify that `b` is always true.

```
bool b = true;

active proctype main() {
    printf("hello world!\n");
    b = false;
}
```

Standard Approach:

- add the LTL formula in `foo.pml`:

  ```
  ltl p1 { [] b }
  ```

- generate, compile and `run` the verifier:

  ```
  ~$ spin -a foo.pml
  ~$ gcc -o pan pan.c
  ~$ ./pan -a -N p1
  ```

  or

  ```
  ~$ spin -search -a -ltl p1 foo.pml
  ```

-a: ask the verifier to also check cyclic executions violating a property

Useful constructs

`_pid`

- unique identifier of a process

`_last`

- pid of the process that performed the last state transition;

`enabled(pid)`

- `true` iff process with identifier `pid` has at least one **executable statement** in its current control state.

Remote References

- allow for inspecting the **local state** of an *active process*:
  - `procname[pid]@label` for **labels**
  - `procname[pid]:varname` for **variables**

Example: (mutual exclusion)

```
ltl p { []! (procname[0]@critical && procname[1]@critical) }
```