

# Towards an Algebraic Specification of Quantum Networks

**Anita Buckley<sup>1</sup>, Pavel Chuprikov<sup>1</sup>, Rodrigo Otoni<sup>1</sup>,**  
**Robert Rand<sup>2</sup>, Robert Soulé<sup>3</sup> and Patrick Eugster<sup>1</sup>**

<sup>1</sup> Università della Svizzera italiana (USI), Switzerland

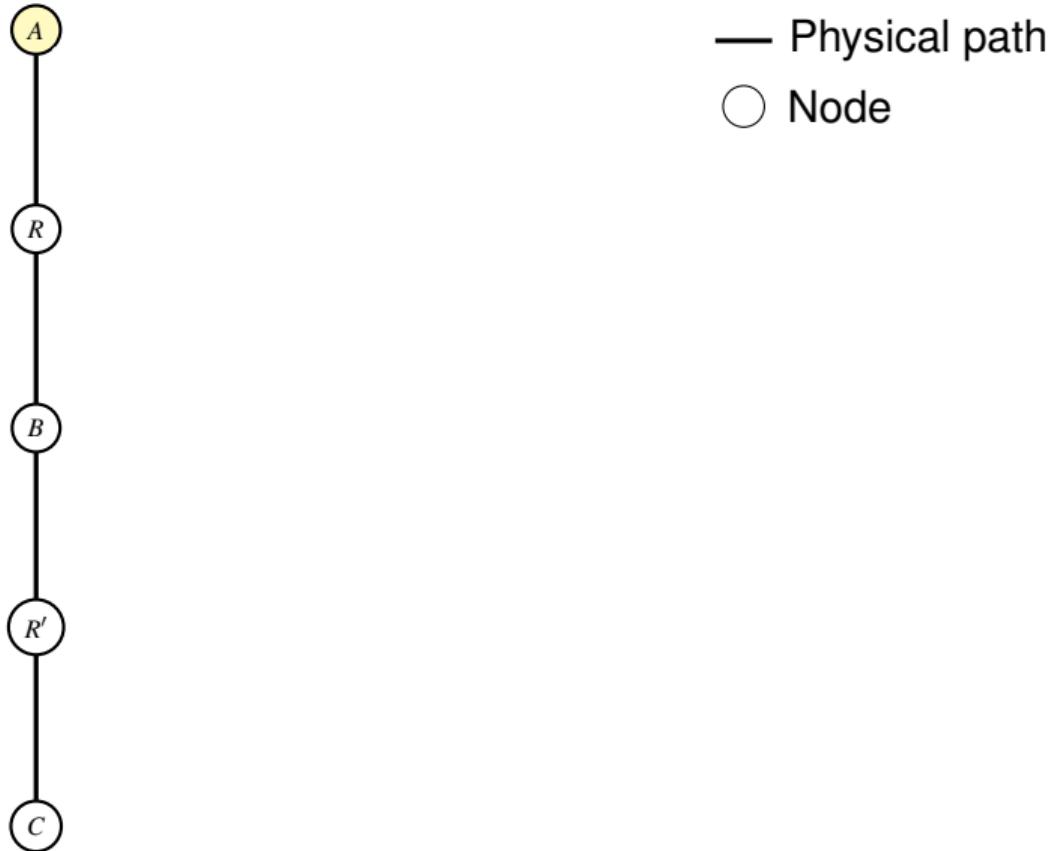
<sup>2</sup> University of Chicago, USA

<sup>3</sup> Yale University, USA

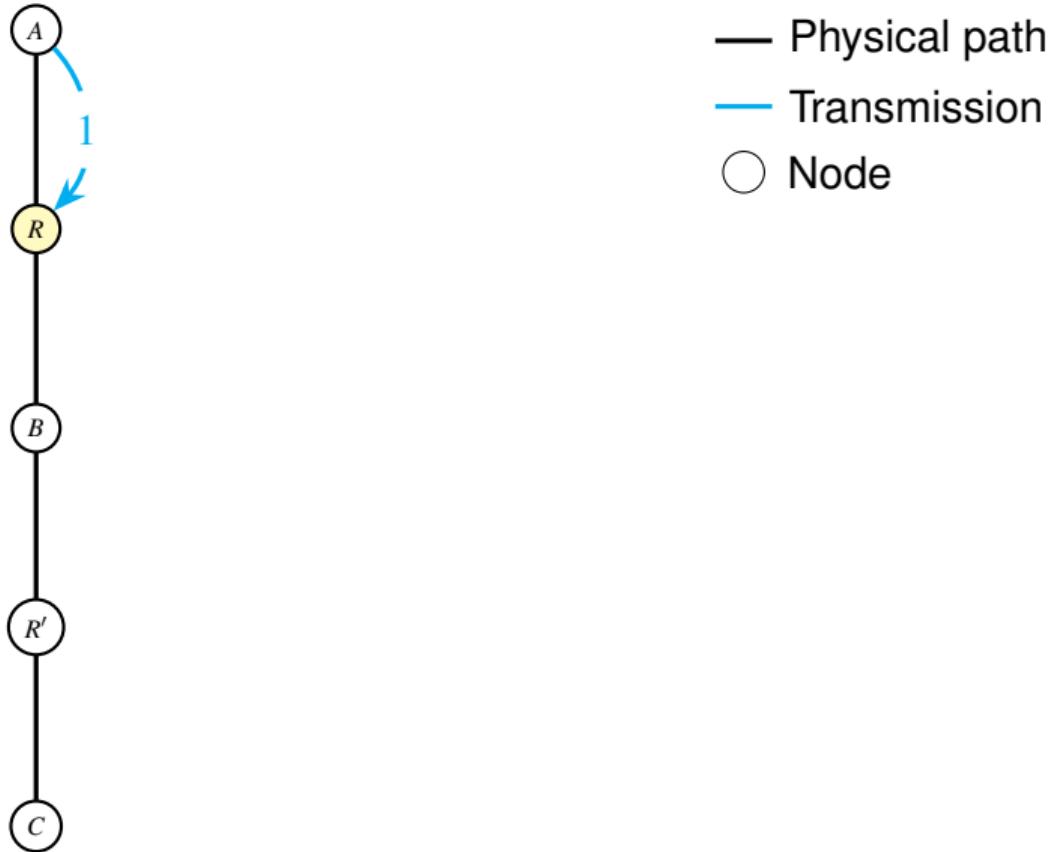
*New York City, 10th September 2023*

1st Workshop on Quantum Networks and Distributed Quantum Computing  
SIGCOMM 2023

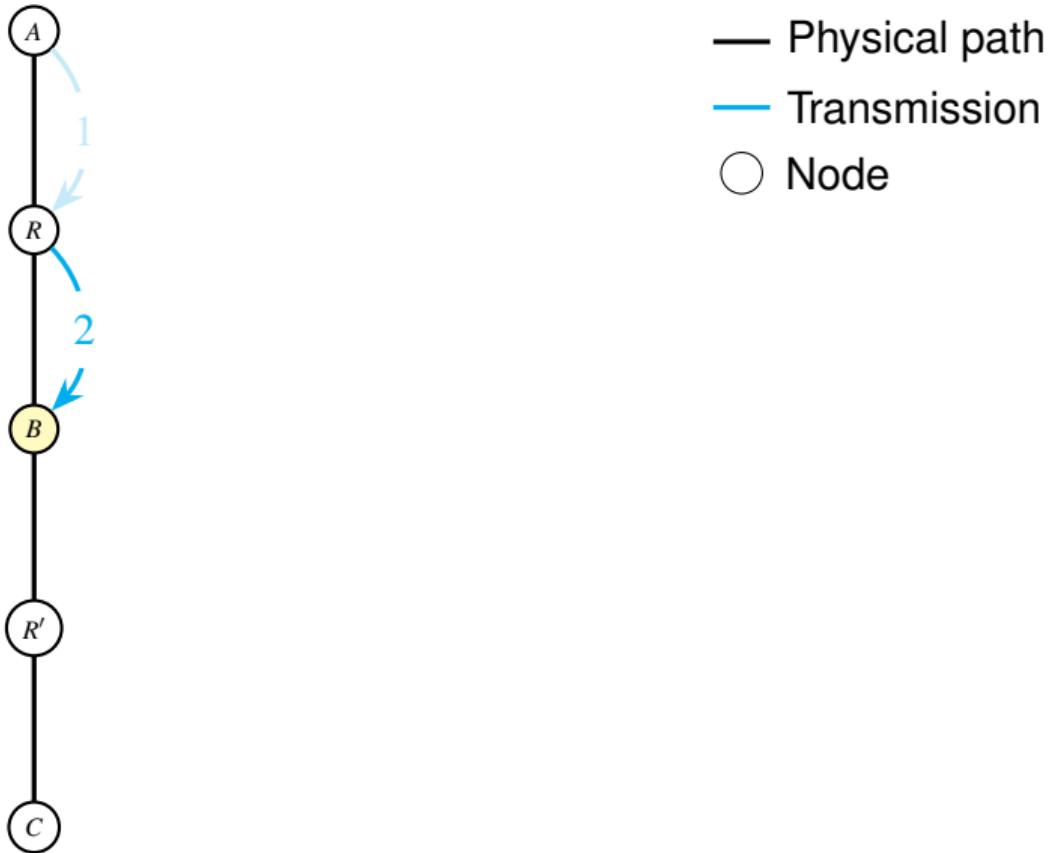
# Classical packet forwarding



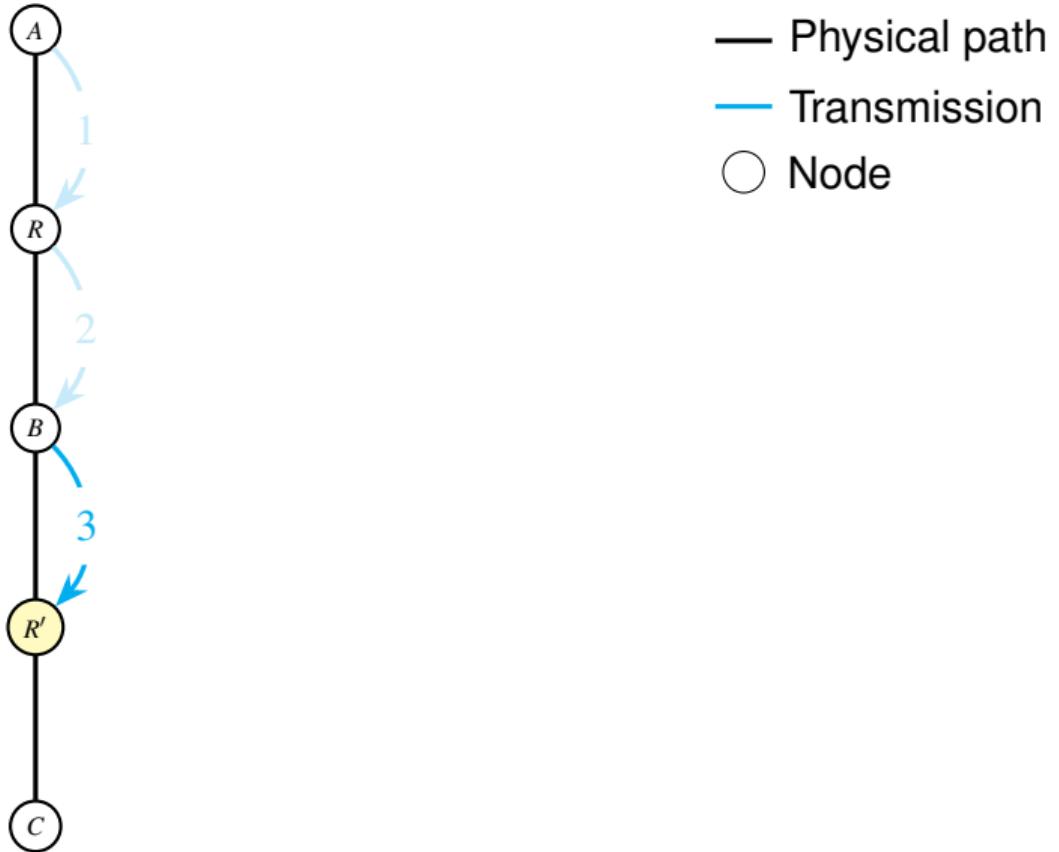
# Classical packet forwarding



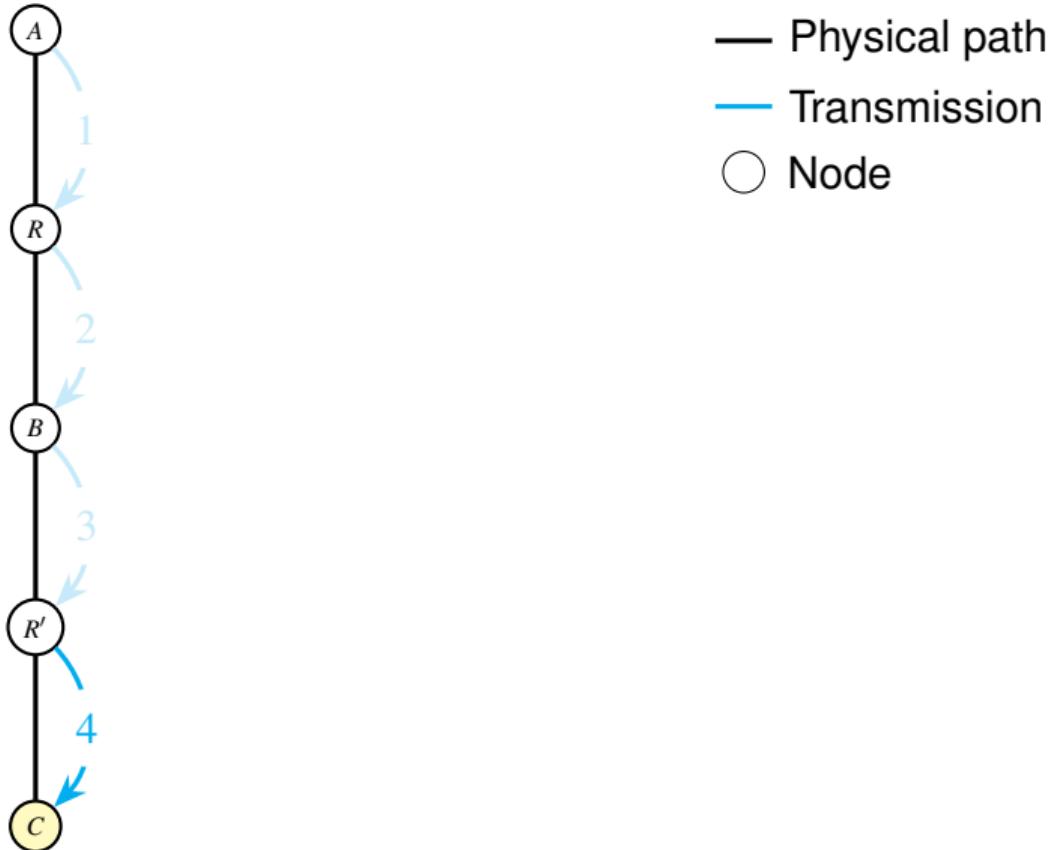
# Classical packet forwarding



# Classical packet forwarding



# Classical packet forwarding



## NetKAT: Semantic Foundations for Networks

Carolyn Jane Anderson  
Swarthmore College \*

Nate Foster  
Cornell University

Arjun Guha  
University of Massachusetts Amherst \*

Jean-Baptiste Jeannin  
Carnegie Mellon University \*

Dexter Kozen  
Cornell University

Cole Schlesinger  
Princeton University

David Walker  
Princeton University

### Abstract

Recent years have seen growing interest in high-level languages for programming networks. But the design of these languages has been largely ad hoc, driven more by the needs of applications and the capabilities of network hardware than by foundational principles. The lack of a semantic foundation left language designers with little guidance in determining how to incorporate new features, and provided no way to reason precisely about their code. This paper presents NetKAT, a new network programming language that is based on a solid mathematical foundation and comes equipped with a sound and complete equational theory. We describe the design of NetKAT, including primitives for filtering, modifying, and transmitting packets; union and sequential composition operators; and a Kleene star operator that handles preambles. We show that NetKAT is the union of a classical and well-known mathematical structure called a Kleene algebra with tests (KAT) and prove that its equational theory is sound and complete with respect to its denotational semantics. Finally, we present practical applications of the equational theory including syntactic techniques for checking readability, proving non-interference properties that ensure isolation between programs, and establishing the correctness of compilation algorithms.

**Categories and Subject Descriptors** D.3.2 [Programming Languages]: Language Classifications—Specialized application languages

**Keywords** Software-defined networking, Fretotic, Network programming languages, Domain-specific languages, Kleene algebra with tests, NetKAT

### 1. Introduction

Traditional network devices have been called ‘the last bastion of mainframe computing’ [9]. Unlike modern computers, which are

\* This work performed at Cornell University.

Permission to make digital or hard copies of all or part of this work for personal use or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for works contained in this volume is held by others than ACM, if specified. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission from [www.acm.org](http://www.acm.org).

POPL '14, January 22–24, 2014, San Diego, CA, USA.  
Copyright © 2014 ACM 978-1-4503-2544-8/14/01...\$15.00  
<http://doi.acm.org/10.1145/2535808.2535822>

implemented with commodity hardware and programmed using standard interfaces, networks have been built the same way since the 1970s: out of special-purpose devices such as routers, switches, firewalls, load balancers, and middle-boxes, each implemented with custom hardware and programmed using proprietary interfaces. This design makes it difficult to extend networks with new functionality and effectively impossible to reason precisely about their behavior.

However, a revolution has taken place with the recent rise of software-defined networking (SDN). In SDN, a general-purpose controller machine manages a collection of programmable switches. The controller responds to network events such as new connections, friend/host topology changes, and shifts in traffic load by reprogramming the switches. Because the controller itself has a global view of the network, it is easy to use SDN to implement a wide variety of standard applications such as shortest-path routing, traffic monitoring, and access control, as well as more sophisticated applications such as load balancing, intrusion detection, and fault-tolerance.

A key benefit of SDN is that it defines open standards that any vendor can implement. For example, the OpenFlow API [21] clearly specifies the capabilities and behavior of switch hardware and defines a low-level language for manipulating their configurations. However, programs written directly for SDN platforms such as OpenFlow are akin to assembly: easy for hardware to implement, but difficult for humans to write.

**Network programming languages.** In recent years, several different research groups have proposed domain-specific languages for SDN [5–7, 23–25, 31, 32]. The goal of these *network programming languages* is to raise the level of abstraction of network programs above hardware-oriented APIs such as OpenFlow, thus enabling them to be easily ported to different SDN applications. For example, the languages developed in the Fretotic project [30] support a two-phase programming model: (i) a general-purpose program responds to network events by generating a static forwarding policy; and (ii) the static policy is compiled and passed to a run-time system that configures the switches using OpenFlow messages. This model balances expressiveness—dynamic policies can be easily added by the general-purpose program—but generates a sequence of static policies for simplicity—forwarding rules are written in a simple domain-specific language with a small semantics, so programs can be analyzed and even verified using automated tools [7, 26].

Still, it has never been clear what features a static policy language should support. The initial version of Fretotic [6] used simple lists of predicate-action rules as policies, where the actions in-

# NetKAT: Semantic Foundations for Networks

Carolyn Jane Anderson  
Swarthmore College<sup>\*</sup>

Nate Foster  
Cornell University

Arijit Guha

Jean-Baptiste Jeannin  
Carnegie Mellon University<sup>\*</sup>

Dexter Kozen  
Cornell University

Cole Schlesinger  
Princeton University

## Abstract

Recent years have seen growing interest in high-level languages for programming networks. But the design of these languages has been largely ad hoc, driven more by the needs of applications and the capabilities of network hardware than by foundational principles. The lack of a semantic foundation left language designers with little guidance in determining how to incorporate new features, and provided no way to reason about the correctness of their code. This paper presents NetKAT, a new network programming language that is based on a solid mathematical foundation and comes equipped with a sound and complete equational theory. We describe the design of NetKAT, including primitives for filtering, modifying, and transmitting packets; union and sequential composition operators; and a Kleene star operator that binds previous programs. We show that NetKAT is the union of a classical and well-known mathematical structure called a Kleene algebra with tests (KAT) and prove that its equational theory is sound and complete with respect to its denotational semantics. Finally, we present practical applications of the equational theory including syntactic predicates for checking readability, proving non-interference properties that ensure isolation between programs, and establishing the correctness of compilation algorithms.

**Categories and Subject Descriptors** D.3.2 [Programming Languages]: Language Classifications—Specialized languages; D.3.2 [Programming Languages]: Language Classifications—Specialized languages.

**Keywords** Software-defined networking, Fretwork, Network programming languages, Domain-specific languages, Kleene algebra with tests, NetKAT.

## 1 Introduction

Traditional network devices have been called “the last bastion of mainframe computing” [9]. Unlike modern computers, which are

<sup>\*</sup>This work performed at Cornell University.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for works contained in this volume is held by their ACM members. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission from [www.acm.org](http://www.acm.org).

POPL '14, January 22–24, 2014, San Diego, CA, USA.  
Copyright © 2014 ACM 978-1-4503-2544-8/14/01...\$15.00  
<http://doi.acm.org/10.1145/2539838.2539842>

## Probabilistic NetKAT

Nate Foster<sup>1</sup>, Dexter Kozen<sup>1</sup>, Konstantinos Mamouras<sup>2\*</sup>,  
Mark Reitblatt<sup>3\*</sup>, and Alexandra Silva<sup>4</sup>

<sup>1</sup> Cornell University

<sup>2</sup> University of Pennsylvania

<sup>3</sup> Facebook

<sup>4</sup> University College London

**Abstract.** This paper presents a new language for network programming based on a probabilistic semantics. We extend the NetKAT language with new primitives for expressing probabilistic behaviors and enrich the semantics from one based on deterministic functions to one based on measurable functions on sets of packet histories. To establish fundamental properties of the language, we prove that it is a complete semantics of the deterministic language, show that it satisfies a number of natural equations, and develop a notion of representation. We present case studies that show how the language can be used to model a diverse collection of scenarios drawn from real-world networks.

## 1 Introduction

Formal specification and verification of networks has become a reality in recent years with the emergence of network-specific programming languages and property-checking tools. Programming languages like Fretwork [1], Pyretic [36], Maple [32], FlowLog [38], and others are enabling programmers to specify the intended behavior of a network in terms of high-level constructs such as Boolean predicates and functions on packets. Verification tools like Header Space Analysis [21], VeriFlow [22], and NetKAT [12] are making it possible to check properties such as safety, liveness, loop freedom, and traffic isolation automatically.

However, despite many notable advances, these frameworks all have a fundamental limitation: they model network behavior in terms of deterministic packet-processing functions. This approach works well enough in settings where the network function is simple, or where the properties of interest only concern the forwarding paths used to carry traffic. But it does not provide satisfactory accounts of more complicated situations that often arise in practice:

- Congestion: the network operator wishes to calculate the expected degree of congestion on each link given a model of the demands for traffic.
- Failures: the network operator wishes to calculate the probability that packets will be delivered to their destination, given that devices and links fail with a certain probability.

<sup>\*</sup>This work performed at Cornell University.



### Abstract

Over the past 5–10 years, the rise of software-defined networking (SDN) has inspired a wide range of new research, theories, heuristics and languages for programming, managing, and debugging systems under behavior. Otherwise, these systems are *ad-hoc*—languages for managing and specifying network policies, or for writing code for network management and debugging. In this paper, we present a solid framework, called Temporal NetKAT, capable of facilitating all of these tasks at once. As its name suggests, Temporal NetKAT is a system of formalized temporal logic (there were linear temporal logic and interval temporal logic) together with Temporal Network KAT, a language for programming to write down concise programs of a packet’s path through a network. Temporal Network KAT provides a programming language to make temporal predicates, and programs can be easily tested or debugged. Operations on that basis, in addition to being used for programming, can be combined again to build new, general, programs based on a formalized theory of LTL, and NetKAT satisfies proofs of behavioral correctness properties. Using new, general, program-based correctness properties, we show that the semantics of temporal programs is sound, complete, and consistent with their semantics. We have

### Temporal NetKAT

Ryan Beckett  
Princeton University, USA  
ryanbeckett@princeton.edu

Michael Greenberg  
Princeton College, USA  
michaelgreenberg@princeton.edu



David Walker  
Princeton University, USA  
dwalker@princeton.edu

### 1. Introduction

In software-defined networking, a general-purpose controller machine, or cluster of machines, manages a collection of switches, programmable switches throughout a network, and open API such as OpenFlow [29]. In order to build reliable SDN networks, one requires a specification of several key components: a platform for programming policies, a mechanism for monitoring and maintaining traffic patterns and switching packets of interests, and mechanisms for SDN verification. Over the last decade, all of these components have been added in a myriad of approaches. For instance, FlowLog [36], Frenetic [33], OpenFlow [12], NetKAT [11], and others have focus on efficient processing of flows, implementing packet-forwarding policies, DREAM [10], OpenSwitch [43], Path Queries [37], other languages for programming packet-forwarding policies, and so on. In addition, there are various network operators who have developed their own specific tools for managing their networks.

In this paper, we propose a simple, new foundation for managing many of these services in a single framework. More specifically, we begin with NetKAT [1], which uses regular expressions to model the semantics of network protocols. Our main contribution is to extend NetKAT with temporal logic, interval temporal logic, and some predicates. We have

implemented w

standard interfa

the 1970s: out c

firewalls, load

with custom h

faces. This de

faultability is

their behavior.

However, i

software-dep

troller machi

The controlle

from h

as program

has a global

ment a wide

routing, trafl

plasticized i

and fault-t

A major

and vendor

clearly spe

and defini

tions. How

as OpenFl

but differe

nt, Frenetic, Network pro-

gramming Lan-

guage, Kleene algebra

lled ‘the last bastion

n’ computers, which are

of this work for personal or

not be distributed or

used by others. ACM

by otherwise, or republish,

reprint and/or a

message

can be

a step

or an

manus

tomat

Si

guar

ple li

of congestion on

on each link

given a mod

of the demands for

traffic.

failures the netw

operator wi

chance that pa

cots will be deliv

er to their desti

nation, given th

at devices and li

ts fall

with a certai

probability.

<sup>1</sup> Work performed at Cornell University.

## NetKAT: Semantic Foundations for Networks

Carolyn Jane Anderson  
Swarthmore College \*

Nate Foster  
Cornell University

Arijit Guha

Jean-Baptiste Jeannin  
Carnegie Mellon University \*

Dexter Kozen  
Cornell University

Cole Schlesinger  
Princeton University

## Probabilistic NetKAT

Nate Foster<sup>1</sup>, Dexter Kozen<sup>1</sup>, Konstantinos Mamouras<sup>2\*</sup>,  
Mark Reitblatt<sup>3\*</sup>, and Alexandra Silva<sup>4</sup>

<sup>1</sup> Cornell University

<sup>2</sup> University of Pennsylvania

<sup>3</sup> Facebook

<sup>4</sup> University College London

**Abstract.** This paper presents a new language for network programming based on a probabilistic semantics. We extend the NetKAT language with primitives for expressing probabilistic behaviors and enrich the semantics from one based on deterministic functions to one based on measurable functions on sets of packet histories to establish fundamental properties of the language. We prove that it is a conservative extension of the deterministic language, show that it is a consequence of statistical equations, and develop a notion of representation. We present case studies that show how the language can be used to model a diverse collection of scenarios drawn from real-world networks.

### 1 Introduction

Formal specification and verification of networks has become a reality in recent years with the emergence of network-specific programming languages and property-checking tools. Programming languages like Frenetic [21], Pyretic [36], Maple [32], FlowLog [38], and others are enabling programmers to specify the intended behavior of a network in terms of high-level constructs such as Boolean predicates and functions on packets. Verification tools like Header Space Analysis [21], VeriFlow [22], and NetKAT [12] are making it possible to check properties such as security, loop freedom, and traffic isolation automatically.

However, despite many notable advances, these frameworks all have a fundamental limitation: they model network behavior in terms of deterministic packet-processing functions. This approach works well enough in settings where the forwarding paths need to carry traffic. But it does not provide satisfactory accounts of more complicated situations that often arise in practice:

- Congestion: the network operator wishes to calculate the expected degree of congestion on each link given a model of the demands for traffic.
- Failures: the network operator wishes to calculate the probability that packets will be delivered to their destination, given that devices and links fail with a certain probability.

<sup>1</sup> Work performed at Cornell University.





## Concurrent NetKAT

Modeling and analyzing stateful, concurrent networks

Jana Wagemaker<sup>1</sup> 26, Nate Foster<sup>2</sup> , Tobias Kappé<sup>3</sup> , Dexter Kozen<sup>4</sup> , Jurijs Ruz<sup>2</sup>, and Alexandra Silva<sup>2</sup>

<sup>1</sup> Radboud University, Nijmegen, The Netherlands  
<sup>2</sup> Carnegie Mellon University, USA  
<sup>3</sup> Cornell University, Ithaca, New York, USA  
<sup>4</sup> ILLC, University of Amsterdam, The Netherlands

**Abstract.** We introduce Concurrent NetKAT (CNetKAT), an extension of NetKAT with operators for specifying and reasoning about concurrent scenarios where multiple paths interact through state. We provide a model of the language based on partially-ordered multiset rewriting (pomesa), which uses a well-established mathematical structure to define the denotational semantics of concurrent languages. We provide a sound and complete axiomatization of this model. Finally, we illustrate the use of CNetKAT through examples. More generally, CNetKAT can be used as an algebraic framework for reasoning about programs with both local state (in packets) and global state (in a global store).

**Keywords:** Concurrent Kleene algebras, NetKAT, completeness, concurrency

## 1 Introduction

Kleene algebras (KA) is a well-studied formalism [20,23,34,8] for analyzing and verifying imperative programs. Over the past few decades, various extensions of KA have been proposed for modeling increasingly sophisticated scenarios. For example, Kleene algebras with tests (KAT) [11] models conditional control flow while NetKAT [3,10] models behavior in packet-switched networks.

A key limitation of NetKAT, however, is that the language is stateless and sequential. It cannot model programs composed in parallel, and it offers no way to reason algebraically about the effects induced by multiple concurrent programs. Meanwhile, the software-defined networking (SDN) paradigm has evolved to include richer functionality based on standard programming including data aggregation and policy routing. In languages like P4 [4], issues of concurrency arise because the semantics depends on the order that packets are processed.

Given this context, it is natural to wonder whether we can add concurrency to NetKAT while retaining the elegance of the underlying framework. In this paper, we answer this question in the affirmative, by developing CNetKAT. However, to do this, we must overcome several challenges. A first hurdle is that networks exhibit many different forms of concurrent behavior. The most obvious source

of concurrency is due to the fact that multiple devices share the same physical hardware. This motivates our focus on distributed systems, and we show that the semantics of CNetKAT can be derived from a general theory of distributed systems.

Given this insight, we turn to the problem of how to specify and verify programs with Test-and-set predicates. After programming to write down concise programs as a packet's path through the network, we must convert to more detailed programs on that basis. To address this challenge, we propose a new, general, programming language for distributed systems, called *Dynamic Networks*. We show that the semantics of CNetKAT follows naturally from this language, and we show that the semantics of CNetKAT can be derived from a general theory of distributed systems.

Buckley et al.

## NetKAT: Semantic Foundations for Networks

Carolyn Jane Anderson  
Swarthmore College \*

Nate Foster  
Cornell University

Arijit Guha

Jean-Baptiste Jeannin  
Carnegie Mellon University \*

Dexter Kozen  
Cornell University

Cole Schlesinger  
Princeton University

### Abstract

Recent years have seen growing interest in high-level languages for programming networks. But the design of these languages has been largely ad hoc, driven more by the needs of applications and the capabilities of network hardware than by foundational principles. The lack of a semantic foundation left language designers with little guidance in determining how to incorporate new features, and programmers without a means to reason precisely about their code.

This paper presents NetKAT, a new network programming language that is based on a solid mathematical foundation and comes equipped with a sound and complete equational theory. We describe primitives for filtering, modifying, and sequential composition operators that iterates programs. We show an axiomatic and well-studied mathematical foundation with tests (KAT) and prove

implemented w standard interfaces the 1970s: out-of-firewalls, load with custom headers. This defines their behavior and their behavior.

However, i

softswitches

the controller

from h

as program

has a globa

ment a wide



David Walker  
University of Wisconsin, USA  
[dwalker.cs.wisc.edu](http://dwalker.cs.wisc.edu)



\*

†

‡

§

¶

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

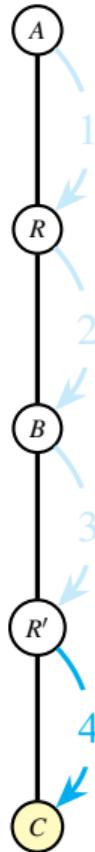
\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

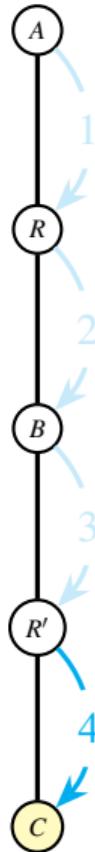
## NetKAT expression $p^*$ is encoding network's behavior



$$p \equiv (\text{SW} = A; \text{SW} \leftarrow R) + (\text{SW} = R; \text{SW} \leftarrow B) + (\text{SW} = B; \text{SW} \leftarrow R') + (\text{SW} = R'; \text{SW} \leftarrow C)$$

$$p^* \equiv 1 + p + p;p + p;p;p + \dots$$

# NetKAT expression $p^*$ is encoding network's behavior

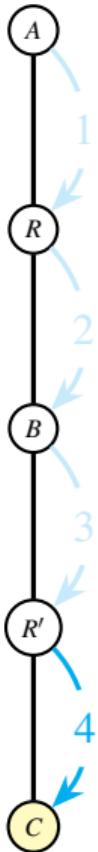


test

$$p \equiv (\text{SW} = A; \text{SW} \leftarrow R) + (\text{SW} = R; \text{SW} \leftarrow B) + (\text{SW} = B; \text{SW} \leftarrow R') + (\text{SW} = R'; \text{SW} \leftarrow C)$$

$$p^* \equiv 1 + p + p;p + p;p;p + \dots$$

NetKAT expression  $p^*$  is encoding network's behavior

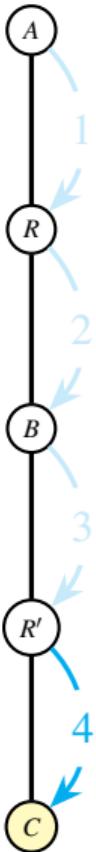


test

action

$$p^* \equiv 1 + p + p; p + p; p; p + \cdots$$

NetKAT expression  $p^*$  is encoding network's behavior



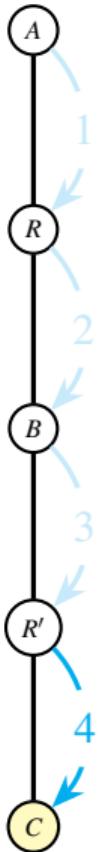
The diagram illustrates the components of a probabilistic model. At the top, two orange rounded rectangles represent the 'test' component on the left and the 'action' component on the right. Below them, four arrows point downwards to a mathematical expression for probability  $p$ . The first arrow from the 'test' component points to the term  $(\text{SW} = A; \text{SW} \leftarrow R)$ . The second arrow from the 'action' component points to the term  $(\text{SW} = R; \text{SW} \leftarrow B)$ . The third arrow from the 'test' component points to the term  $(\text{SW} = B; \text{SW} \leftarrow R')$ . The fourth arrow from the 'action' component points to the term  $(\text{SW} = R'; \text{SW} \leftarrow C)$ .

$$p \equiv (\text{SW} = A; \text{SW} \leftarrow R) + (\text{SW} = R; \text{SW} \leftarrow B) + (\text{SW} = B; \text{SW} \leftarrow R') + (\text{SW} = R'; \text{SW} \leftarrow C)$$

$$p^* \equiv 1 + p + p; p + p; p; p + \cdots$$

## sequential composition

# NetKAT expression $p^*$ is encoding network's behavior



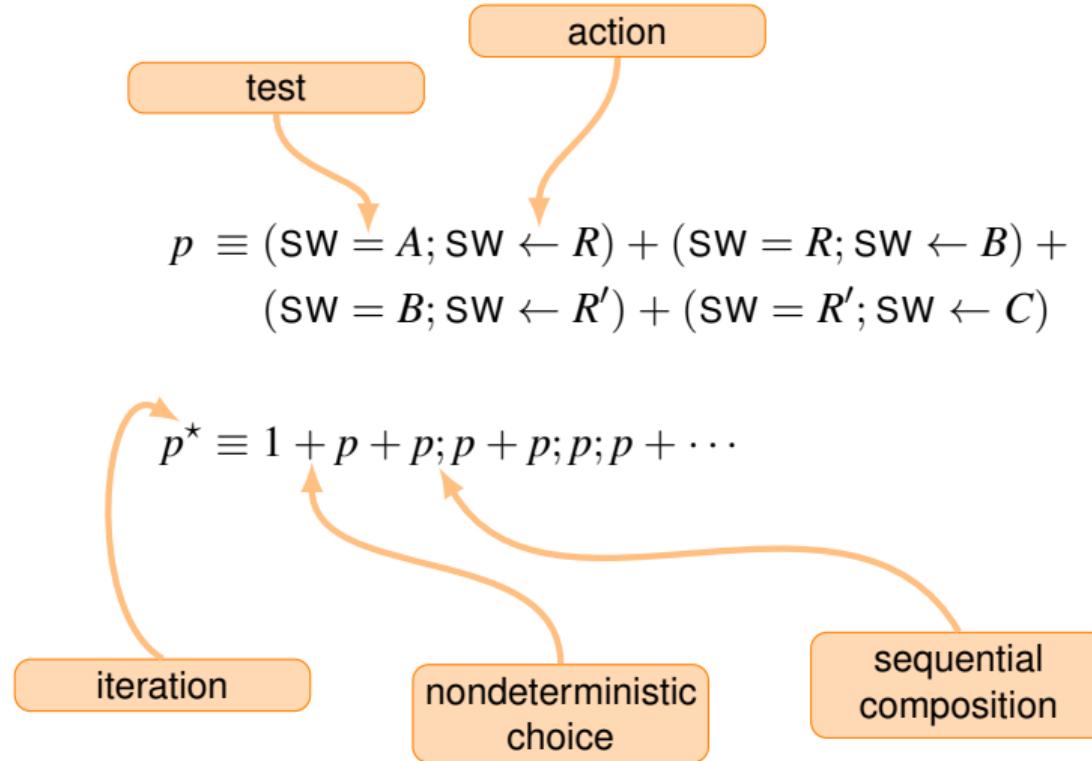
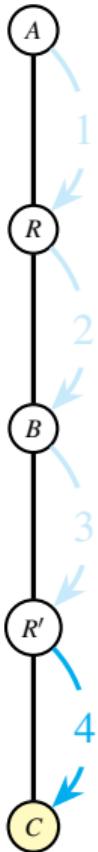
action  
test  
 $p \equiv (\text{SW} = A; \text{SW} \leftarrow R) + (\text{SW} = R; \text{SW} \leftarrow B) + (\text{SW} = B; \text{SW} \leftarrow R') + (\text{SW} = R'; \text{SW} \leftarrow C)$

$$p^* \equiv 1 + p + p; p + p; p; p + \dots$$

nondeterministic choice

sequential composition

# NetKAT expression $p^*$ is encoding network's behavior



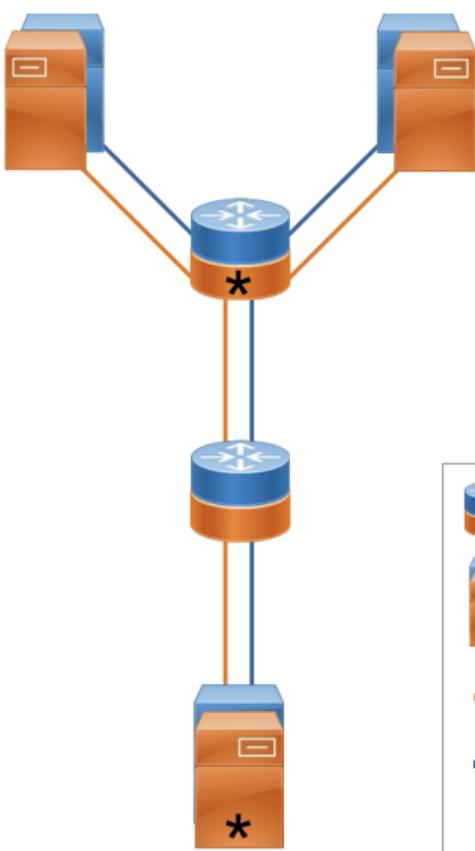


'spooky action at a distance'

# Bell pair: maximally entangled quantum bits

- fundamental unit in quantum networks
- consists of two quantum bits (qubits):  $R \sim B$  is a Bell pair distributed between nodes  $R$  and  $B$
- no headers: control information needs to be sent via separate classical channels





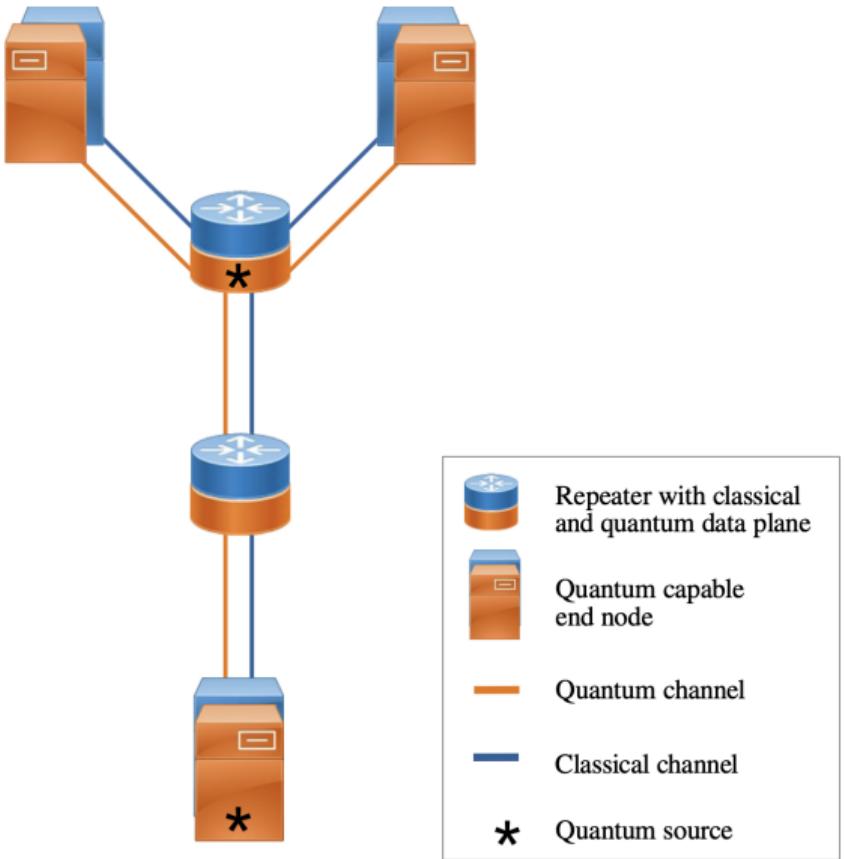
**Quantum network:**  
providing *communication services* to distributed quantum applications

1

	Repeater with classical and quantum data plane
	Quantum capable end node
— orange	Quantum channel
— blue	Classical channel
*	Quantum source

<sup>1</sup>W. Kozlowski and S. Wehner: *Towards Large-Scale Quantum Networks*. NANOCOM (2019)

<sup>2</sup>IRTF, QIRG: *Architectural Principles for a Quantum Internet*. RFC 9340 (2023)



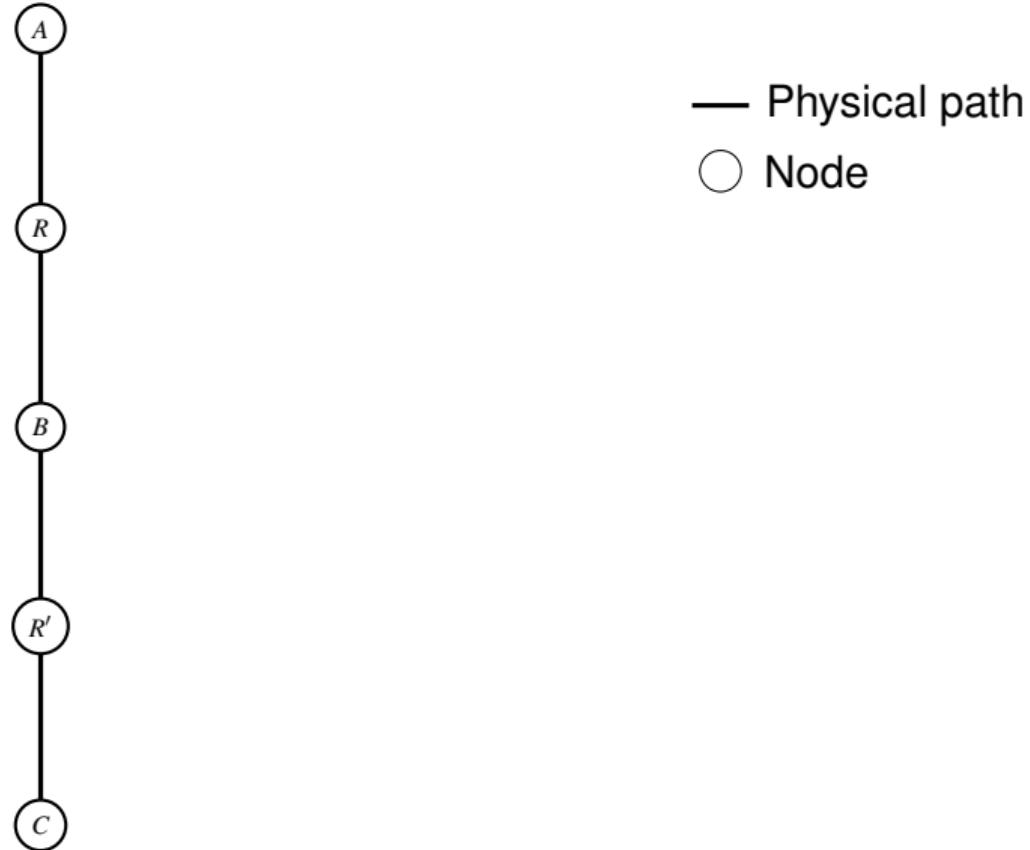
**Quantum network:**  
providing *communication services* to distributed quantum applications

end-to-end Bell pair distribution<sup>2</sup>

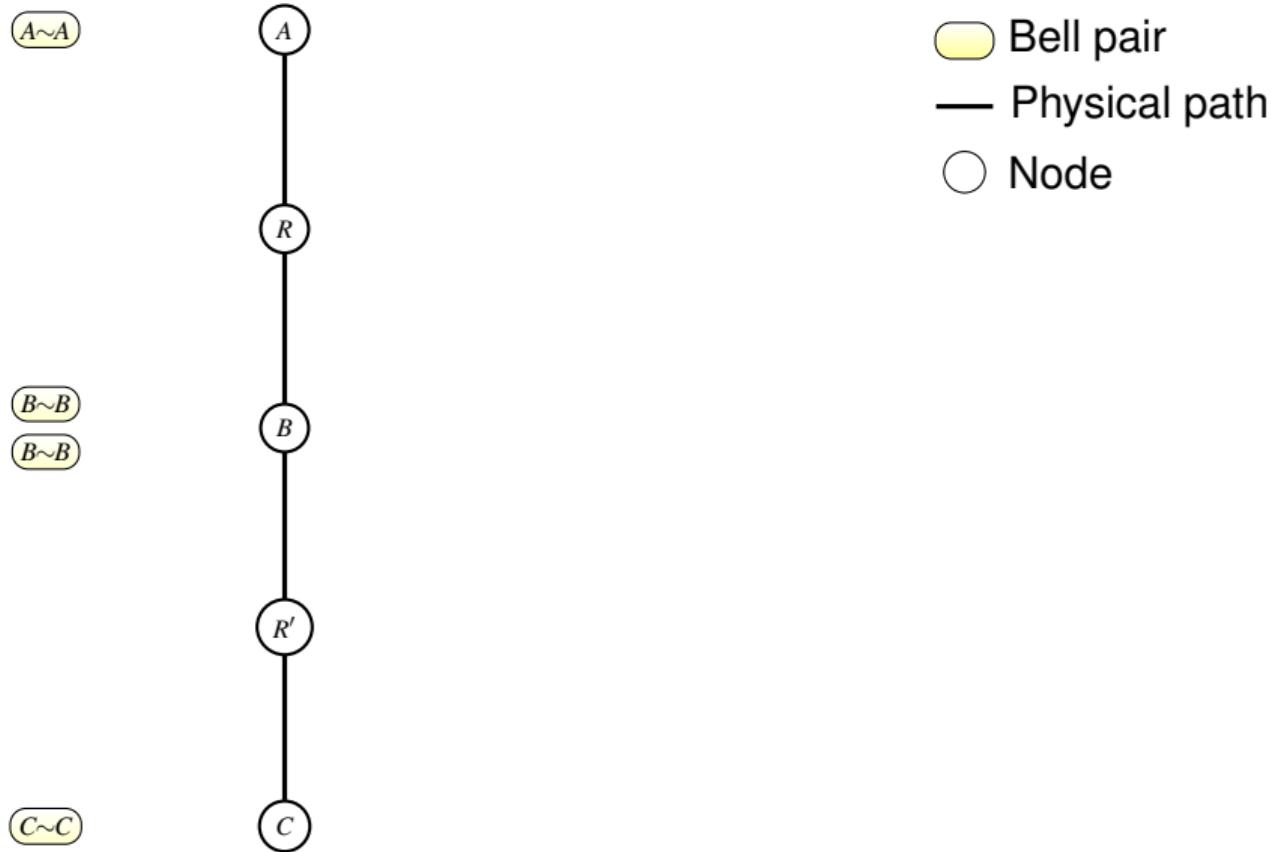
<sup>1</sup>W. Kozlowski and S. Wehner: *Towards Large-Scale Quantum Networks*. NANOCOM (2019)

<sup>2</sup>IRTF, QIRG: *Architectural Principles for a Quantum Internet*. RFC 9340 (2023)

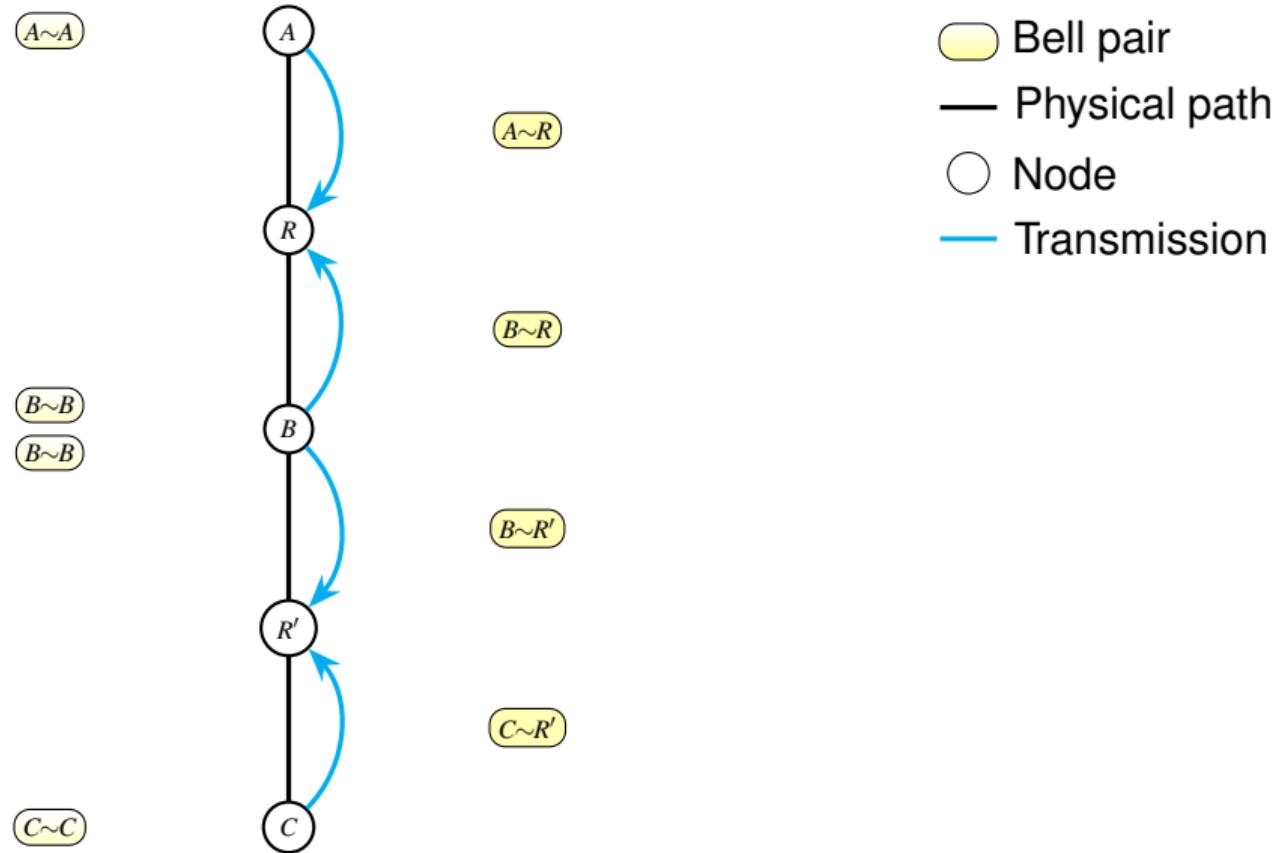
## Bell pair generation



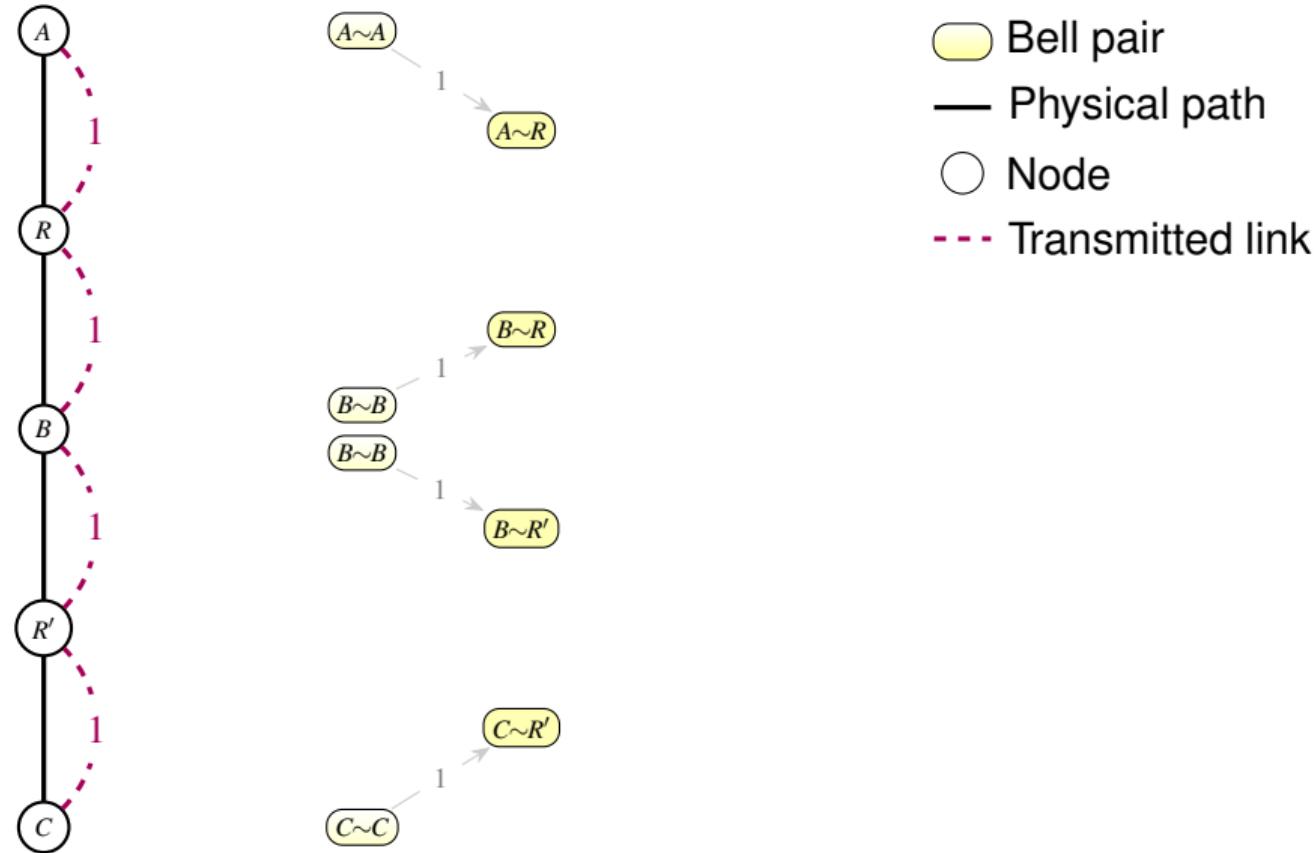
# Bell pair generation: Protocol I



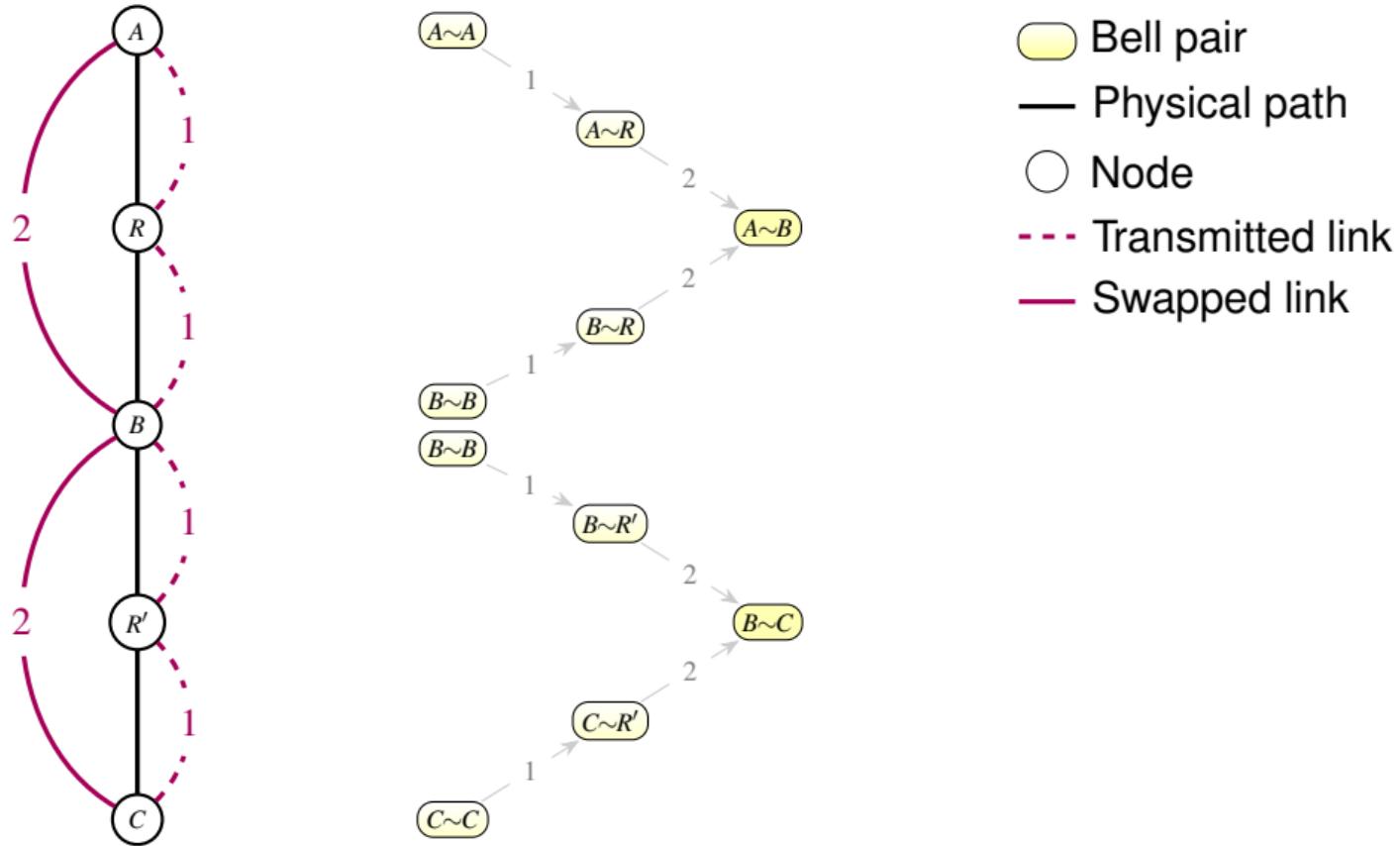
# Bell pair generation: Protocol I



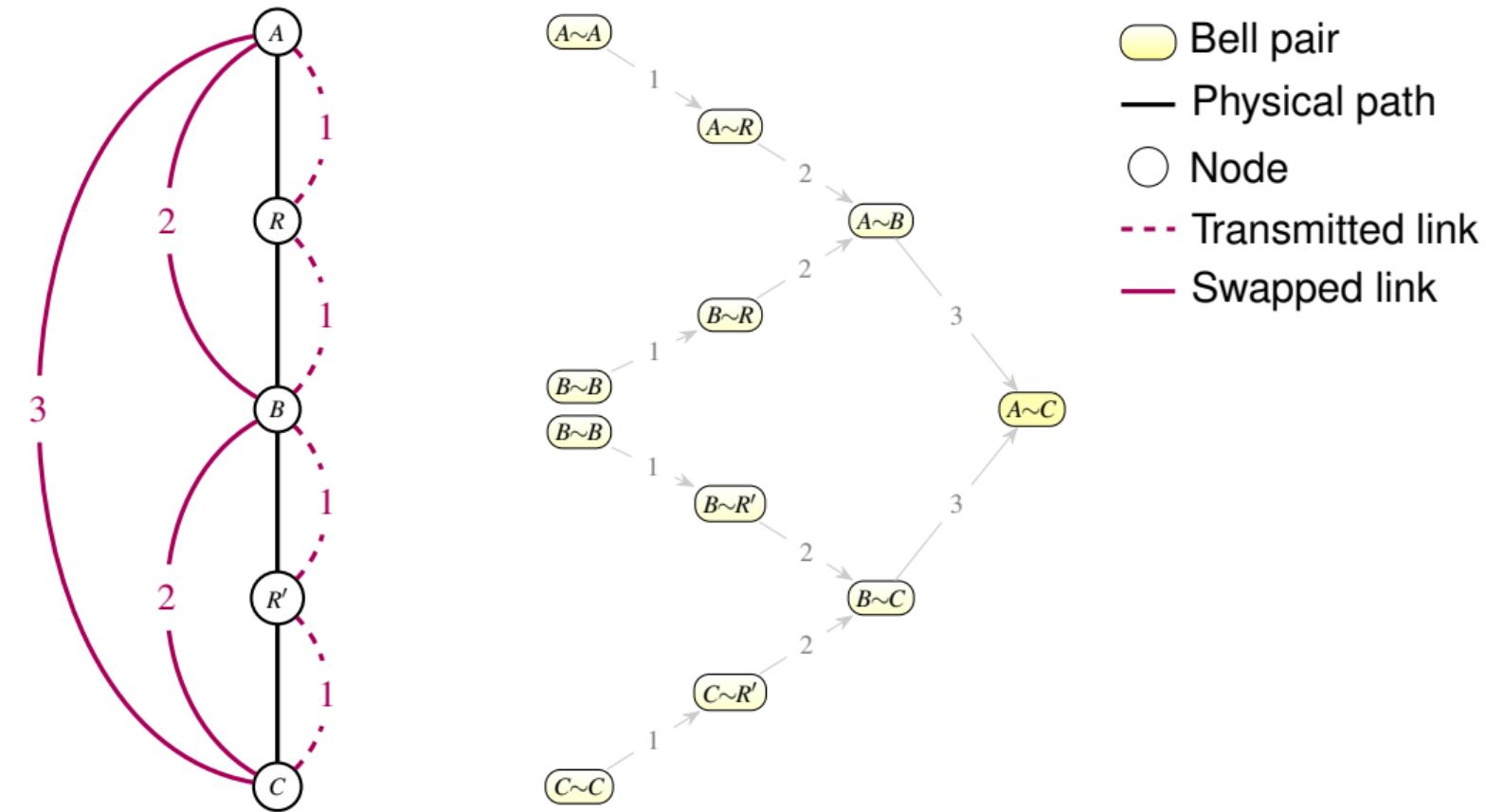
## Bell pair generation: Protocol I, round 1



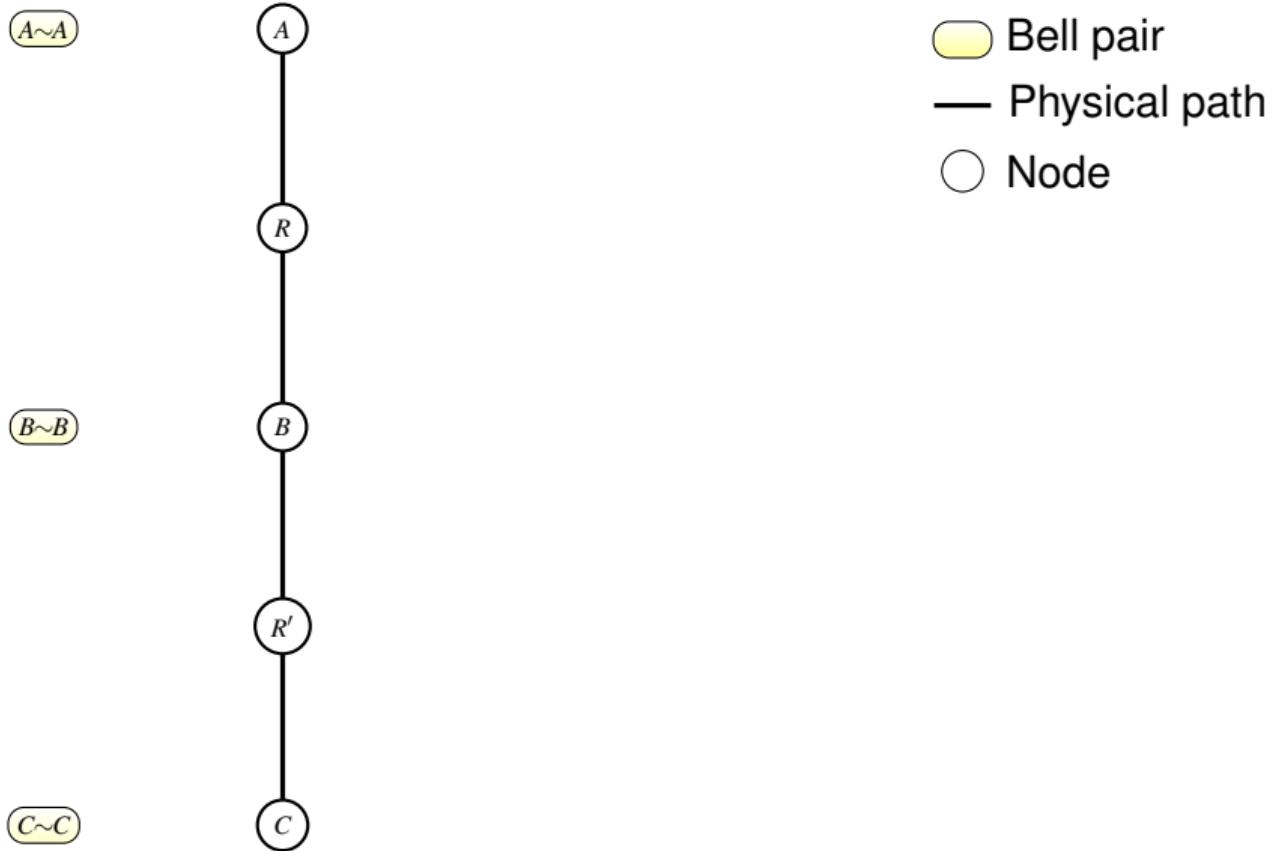
## Bell pair generation: Protocol I, round 2



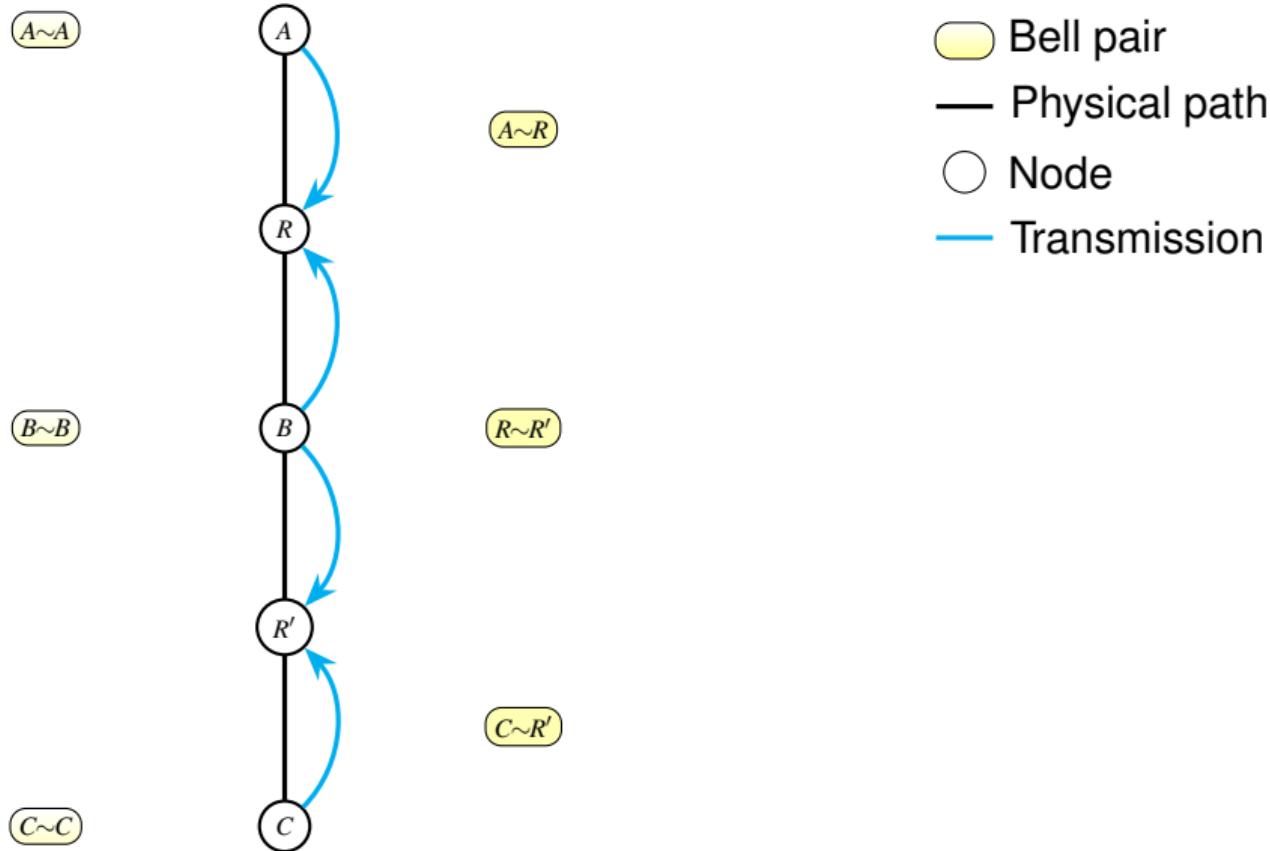
## Bell pair generation: Protocol I, round 3



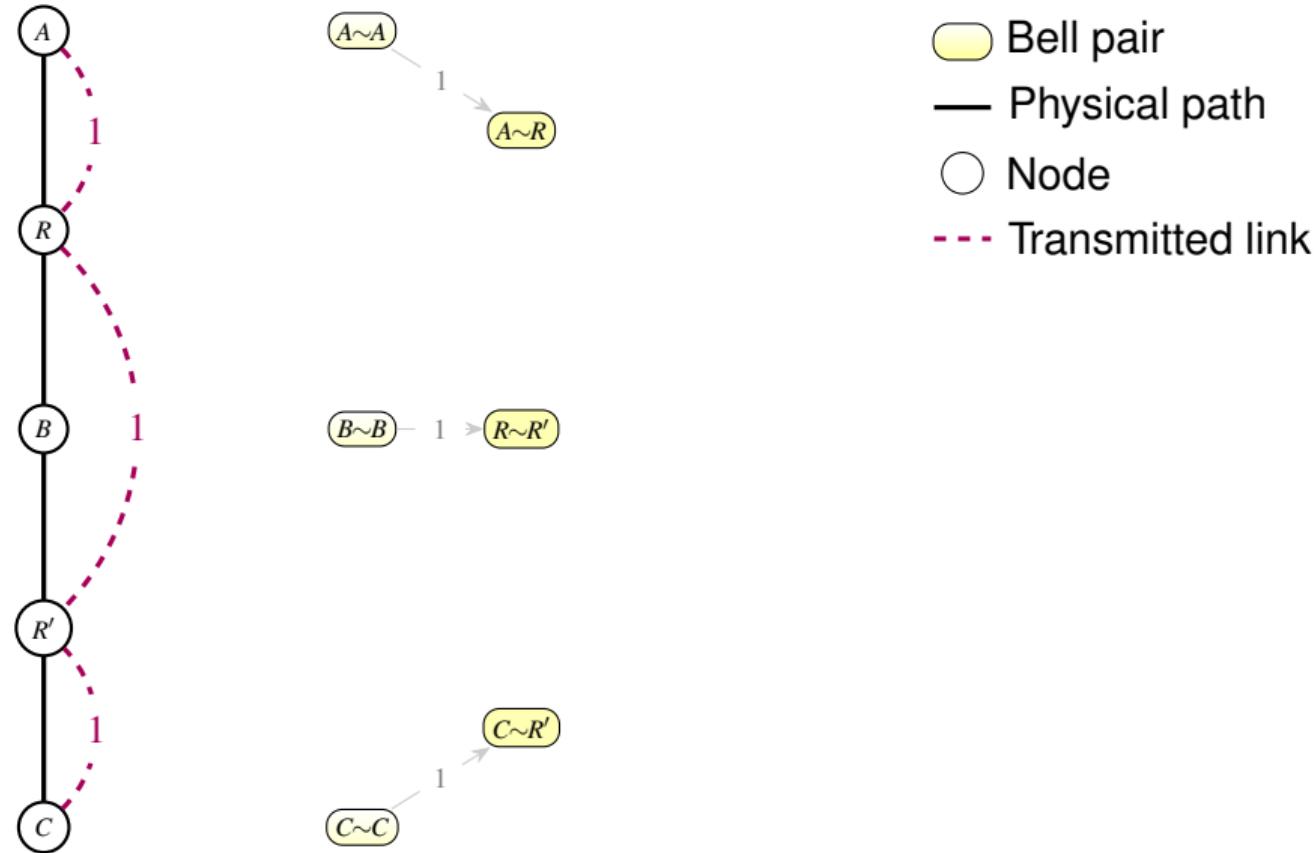
## Bell pair generation: Protocol II



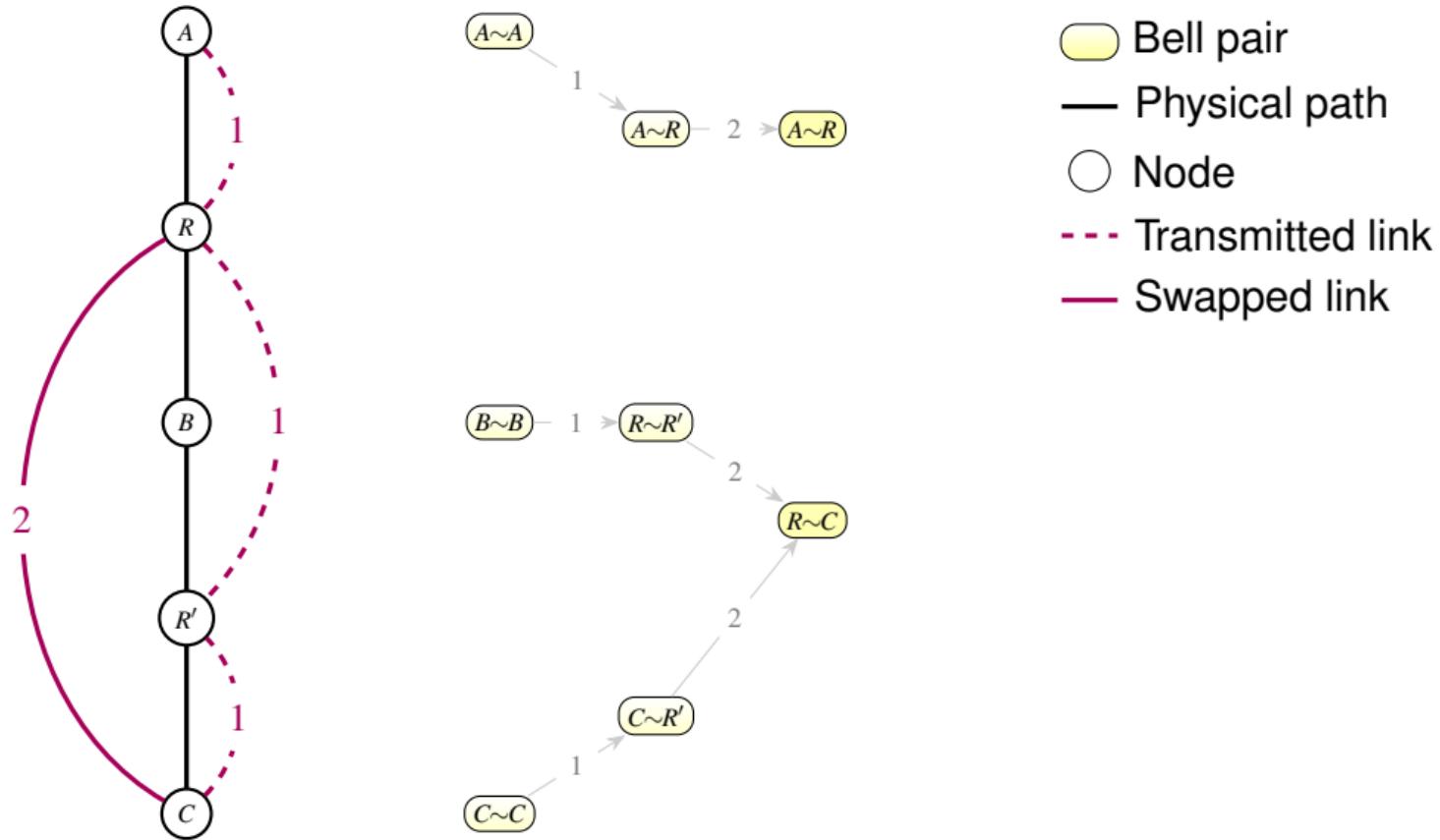
## Bell pair generation: Protocol II



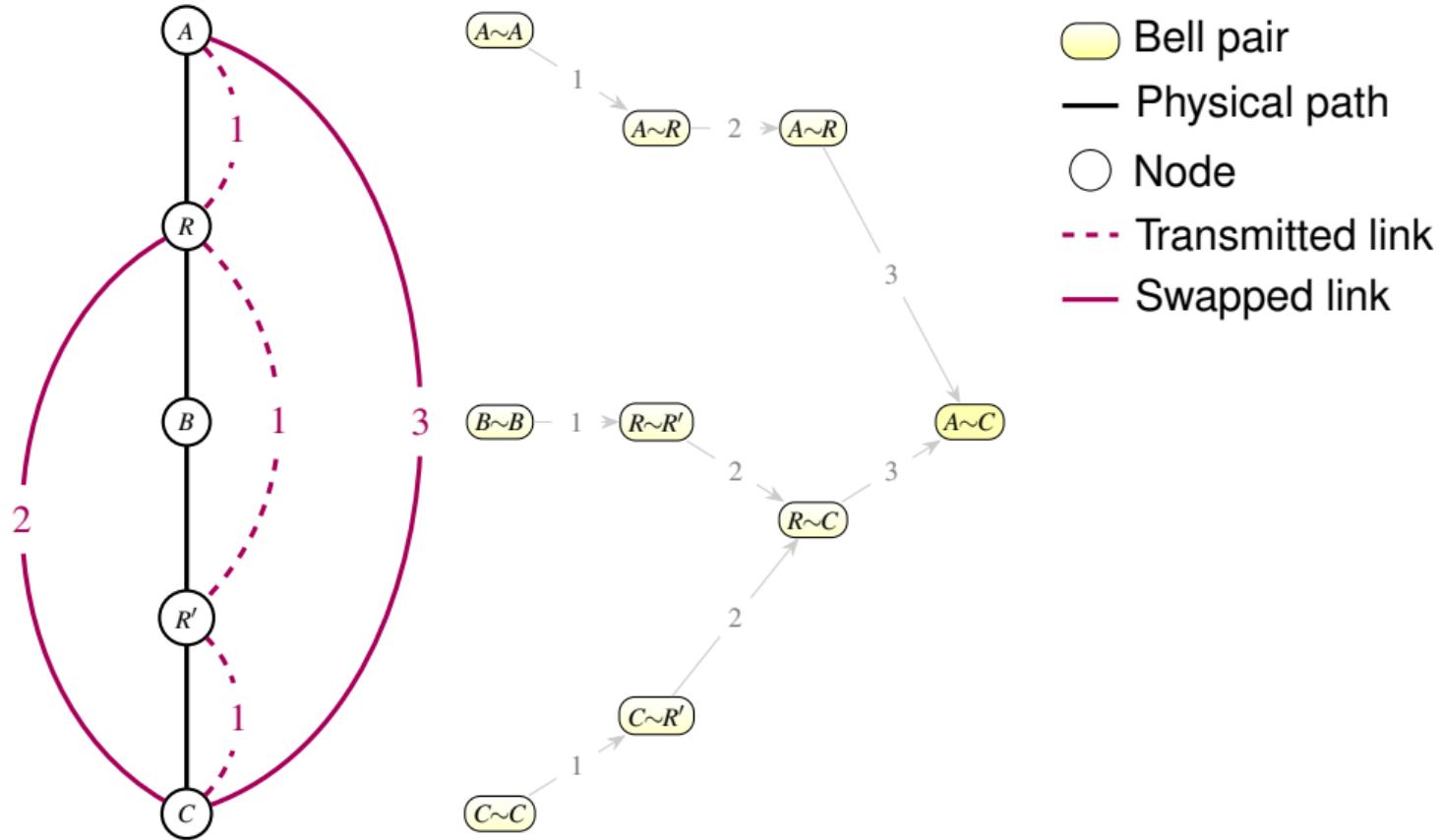
## Bell pair generation: Protocol II, round 1



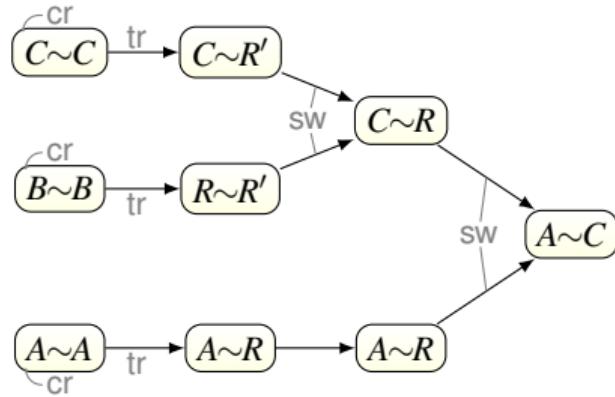
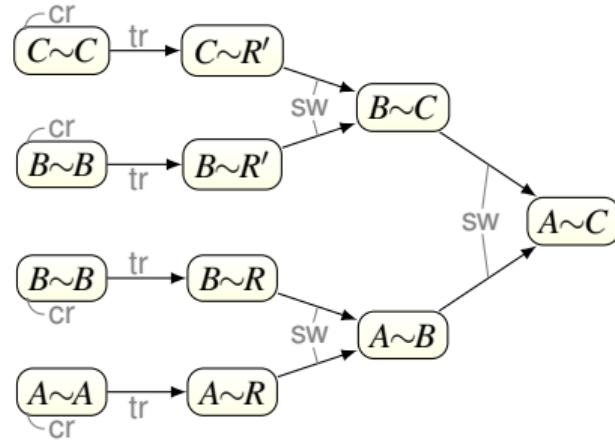
## Bell pair generation: Protocol II, round 2



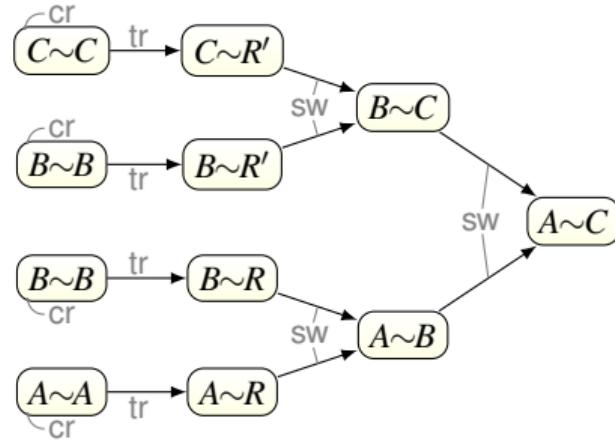
## Bell pair generation: Protocol II, round 3



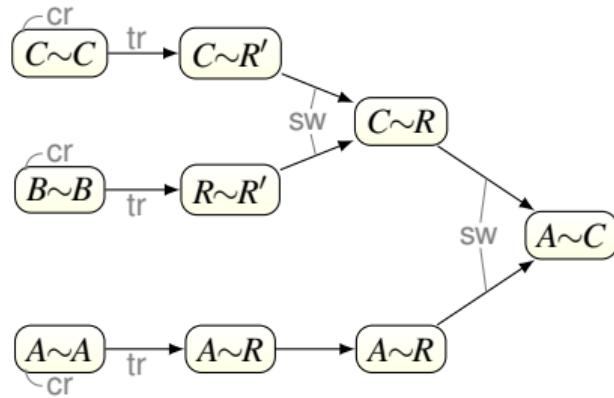
# Specification: Protocol I and Protocol II generating $A \sim C$



# Specification: Protocol I and Protocol II generating $A \sim C$



$(\text{cr}\langle A \rangle \parallel \text{cr}\langle B \rangle \parallel \text{cr}\langle B \rangle \parallel \text{cr}\langle C \rangle);$   
 $(\text{tr}\langle A \rightarrow A \sim R \rangle \parallel \text{tr}\langle B \rightarrow B \sim R \rangle \parallel \text{tr}\langle B \rightarrow B \sim R' \rangle \parallel \text{tr}\langle C \rightarrow C \sim R' \rangle);$   
 $(\text{sw}\langle A \sim B @ R \rangle \parallel \text{sw}\langle B \sim C @ R' \rangle);$   
 $\text{sw}\langle A \sim C @ B \rangle$



$(\text{cr}\langle A \rangle \parallel \text{cr}\langle B \rangle \parallel \text{cr}\langle C \rangle);$   
 $(\text{tr}\langle A \rightarrow A \sim R \rangle \parallel \text{tr}\langle B \rightarrow R \sim R' \rangle \parallel \text{tr}\langle C \rightarrow C \sim R' \rangle);$   
 $\text{sw}\langle C \sim R @ R' \rangle;$   
 $\text{sw}\langle A \sim C @ R \rangle$

## Basic action $r \triangleright o$

$$r \triangleright o: \mathcal{M}(\text{BP}) \rightarrow \mathcal{M}(\text{BP}) \times \mathcal{M}(\text{BP})$$

$$a \mapsto \begin{cases} \textcolor{green}{o} \bowtie \textcolor{blue}{a \setminus r} & \text{if } \textcolor{red}{r} \subseteq a \\ \emptyset \bowtie \textcolor{blue}{a} & \text{otherwise} \end{cases}$$

---

$\bowtie$ : pair of multisets of Bell pairs

## Basic action $r \triangleright o$

required  
Bell pairs

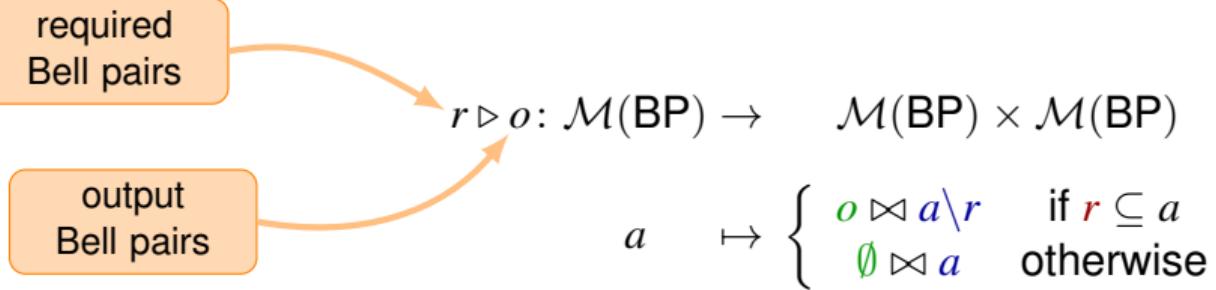
$$r \triangleright o: \mathcal{M}(\text{BP}) \rightarrow \mathcal{M}(\text{BP}) \times \mathcal{M}(\text{BP})$$

$$a \mapsto \begin{cases} o \bowtie a \setminus r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}$$

---

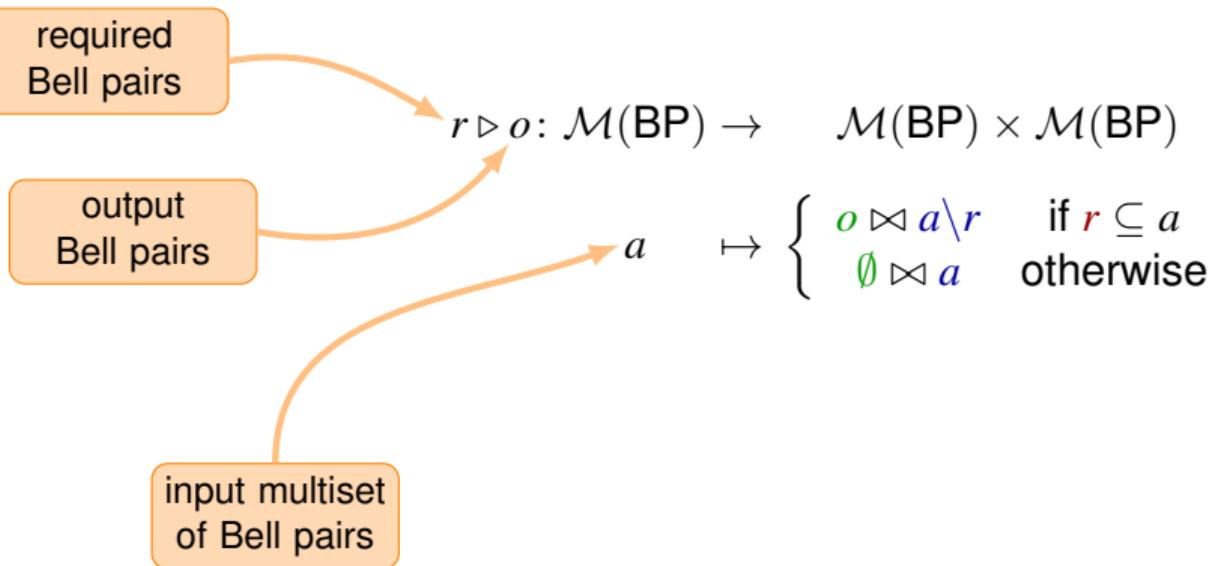
$\bowtie$ : pair of multisets of Bell pairs

## Basic action $r \triangleright o$



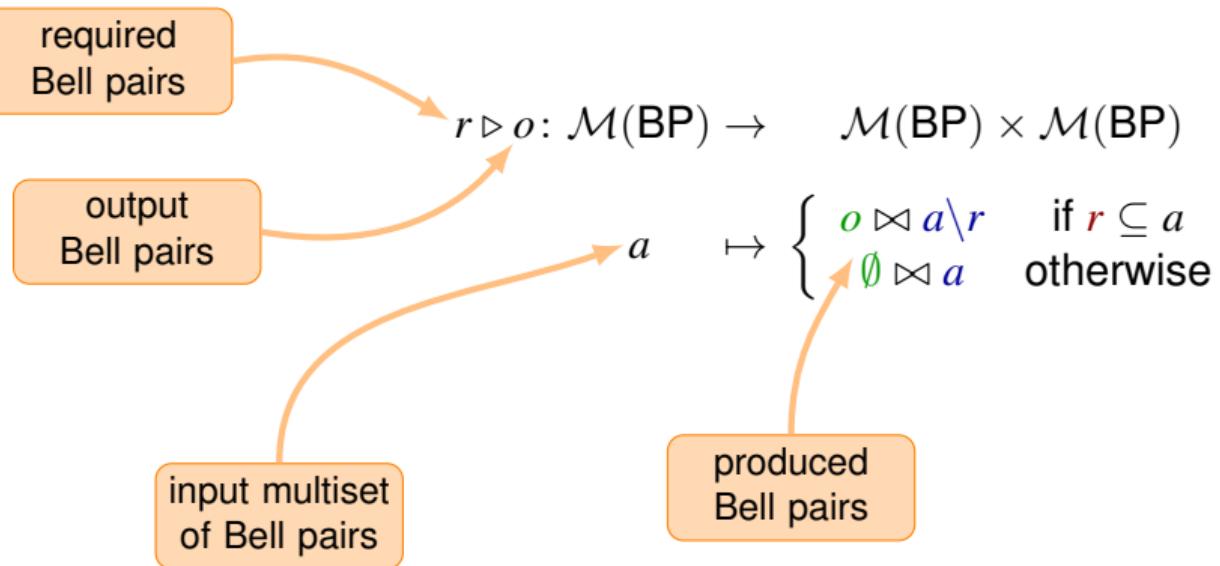
$\bowtie$ : pair of multisets of Bell pairs

## Basic action $r \triangleright o$



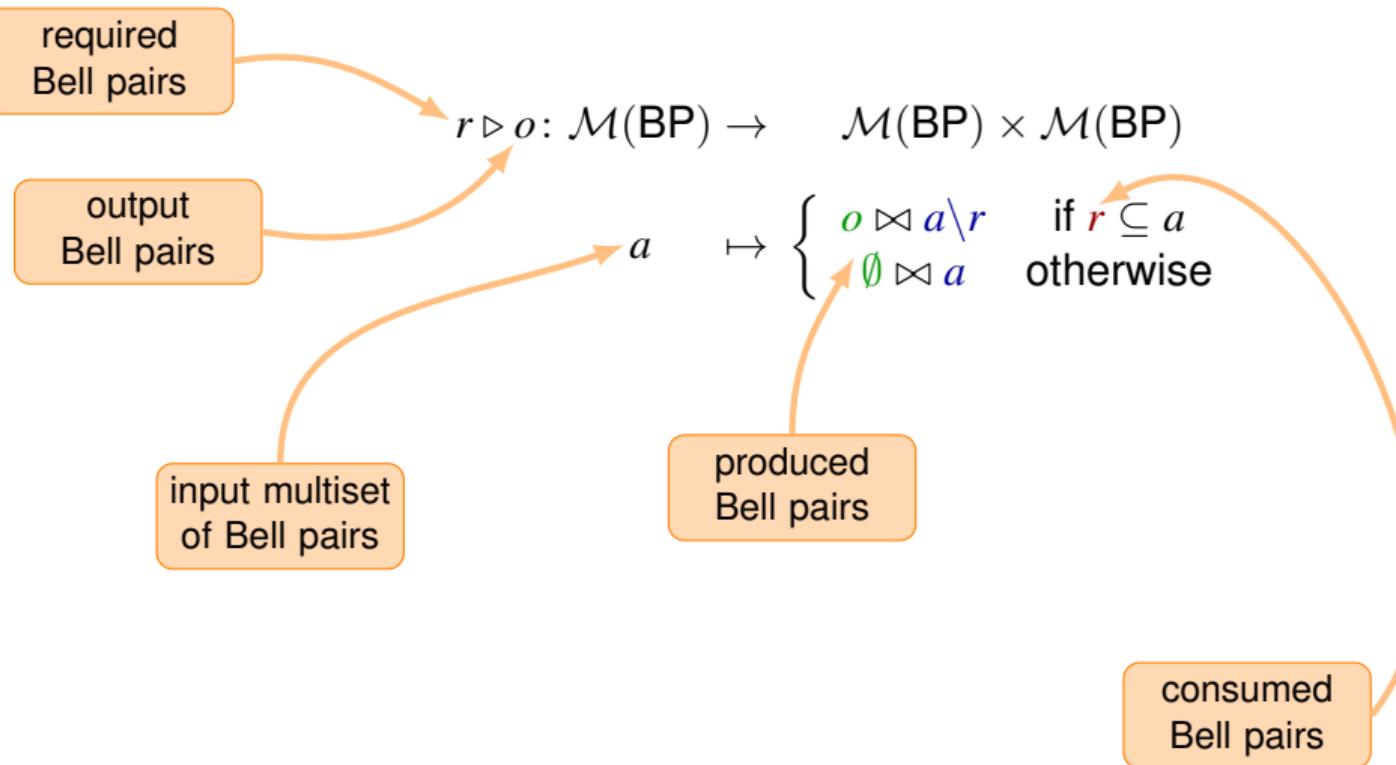
$\bowtie$ : pair of multisets of Bell pairs

## Basic action $r \triangleright o$



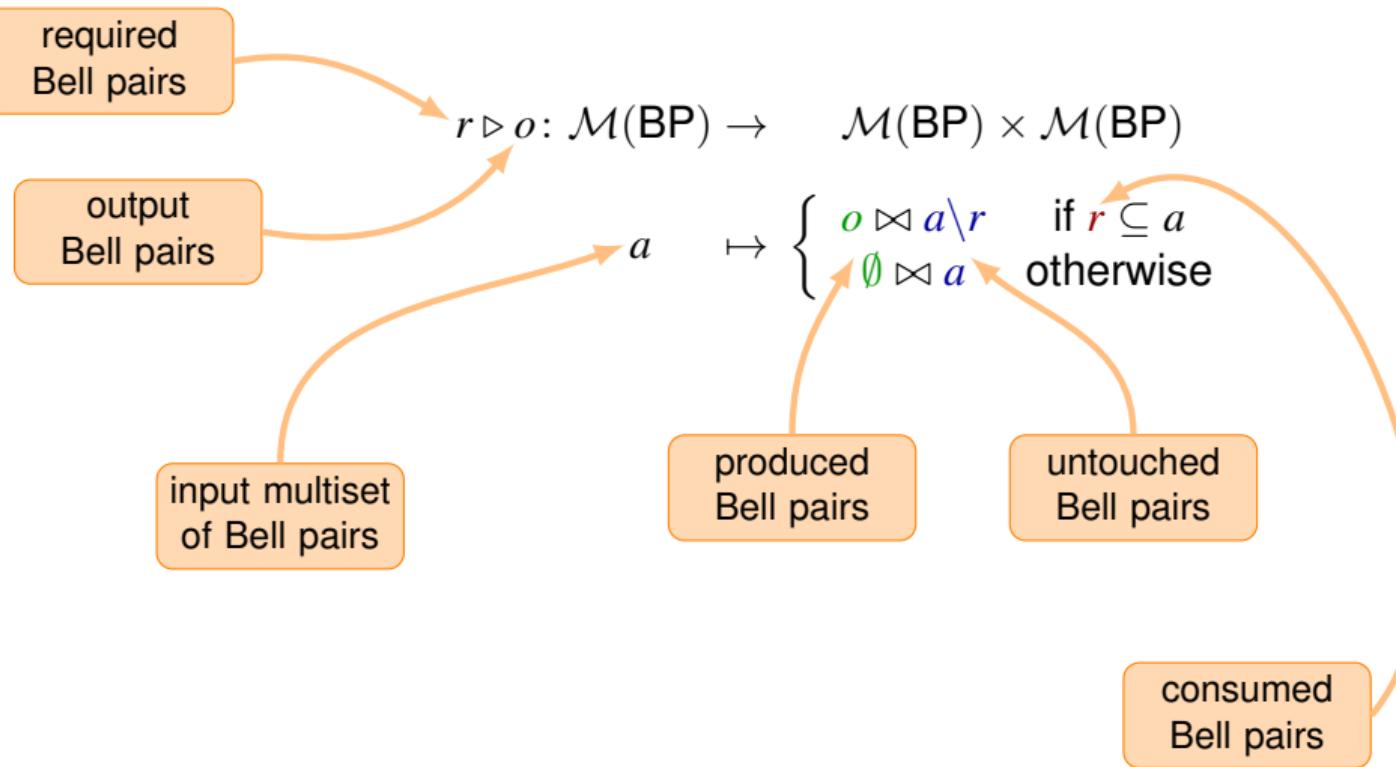
$\bowtie$ : pair of multisets of Bell pairs

## Basic action $r \triangleright o$



$\bowtie$ : pair of multisets of Bell pairs

## Basic action $r \triangleright o$



$\bowtie$ : pair of multisets of Bell pairs

# Basic action

$$r \triangleright o : \mathcal{M}(\text{BP}) \rightarrow \mathcal{M}(\text{BP}) \times \mathcal{M}(\text{BP})$$
$$a \quad \mapsto \begin{cases} o \bowtie a \setminus r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}$$

## Swap action $\text{sw}\langle A \sim B @ R \rangle$

$\{\!\{A \sim R, B \sim R\}\!\} \triangleright \{\!\{A \sim B\}\!\}$  acting on  $a = \{\!\{A \sim R, A \sim R, B \sim R, B \sim R, A \sim B\}\!\}$

$A \sim R$      $B \sim R$      $A \sim R$      $B \sim R$      $A \sim B$

---

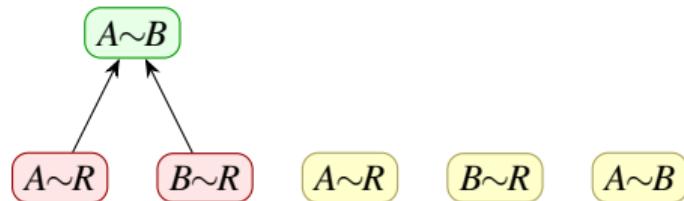
### Input Bell pairs

# Basic action

$$r \triangleright o : \mathcal{M}(\text{BP}) \rightarrow \mathcal{M}(\text{BP}) \times \mathcal{M}(\text{BP})$$
$$a \mapsto \begin{cases} o \bowtie a \setminus r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}$$

## Swap action $\text{sw}\langle A \sim B @ R \rangle$

$\{\!A\!\sim\!R, B\!\sim\!R\}\! \triangleright \{\!A\!\sim\!B\}$  acting on  $a = \{A\!\sim\!R, A\!\sim\!R, B\!\sim\!R, B\!\sim\!R, A\!\sim\!B\}$



Bell pairs: input and consumed, produced

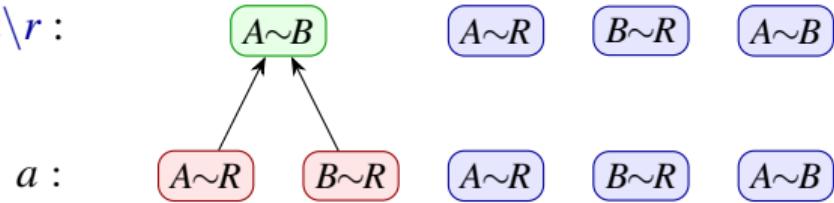
# Basic action

$$r \triangleright o : \mathcal{M}(\text{BP}) \rightarrow \mathcal{M}(\text{BP}) \times \mathcal{M}(\text{BP})$$
$$a \quad \mapsto \begin{cases} o \bowtie a \setminus r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}$$

## Swap action $\text{sw}\langle A \sim B @ R \rangle$

$\{\{A \sim R, B \sim R\}\} \triangleright \{\{A \sim B\}\}$  acting on  $a = \{\{A \sim R, A \sim R, B \sim R, B \sim R, A \sim B\}\}$

$o \bowtie a \setminus r :$



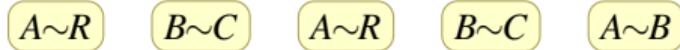
Bell pairs: consumed, produced and untouched

## Basic action

$$r \triangleright o : \mathcal{M}(\text{BP}) \rightarrow \mathcal{M}(\text{BP}) \times \mathcal{M}(\text{BP})$$
$$a \quad \mapsto \begin{cases} o \bowtie a \setminus r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}$$

### Swap action $\text{sw}\langle A \sim B @ R \rangle$

$\{\!\{A \sim R, B \sim R\}\!\} \triangleright \{\!\{A \sim B\}\!\}$  trying to act on  $a = \{\!\{A \sim R, A \sim R, B \sim C, B \sim C, A \sim B\}\!\}$



---

### Input Bell pairs

# Basic action

$$r \triangleright o : \mathcal{M}(\text{BP}) \rightarrow \mathcal{M}(\text{BP}) \times \mathcal{M}(\text{BP})$$
$$a \quad \mapsto \begin{cases} o \bowtie a \setminus r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}$$

## Swap action $\text{sw}\langle A \sim B @ R \rangle$

$\{\!\{A \sim R, B \sim R\}\!\} \triangleright \{\!\{A \sim B\}\!\}$  trying to act on  $a = \{A \sim R, A \sim R, B \sim C, B \sim C, A \sim B\}$



$A \sim R$     $B \sim C$     $A \sim R$     $B \sim C$     $A \sim B$

## Input Bell pairs

## Basic action

$$r \triangleright o : \mathcal{M}(\text{BP}) \rightarrow \mathcal{M}(\text{BP}) \times \mathcal{M}(\text{BP})$$
$$a \quad \mapsto \begin{cases} o \bowtie a \setminus r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}$$

### Swap action $\text{sw}\langle A \sim B @ R \rangle$

$\{\!\{A \sim R, B \sim R\}\!} \triangleright \{\!\{A \sim B\}\!$  trying to act on  $a = \{\!\{A \sim R, A \sim R, B \sim C, B \sim C, A \sim B\}\!$

$$\emptyset \bowtie a : \quad \boxed{A \sim R} \quad \boxed{B \sim C} \quad \boxed{A \sim R} \quad \boxed{B \sim C} \quad \boxed{A \sim B}$$

$$a : \quad \boxed{A \sim R} \quad \boxed{B \sim C} \quad \boxed{A \sim R} \quad \boxed{B \sim C} \quad \boxed{A \sim B}$$

Bell pairs: consumed, produced and untouched

## Basic actions

swap	$\text{sw}\langle A \sim B @ C \rangle \triangleq \{\{A \sim C, B \sim C\} \triangleright \{A \sim B\}$
transmit	$\text{tr}\langle A \rightarrow B \sim C \rangle \triangleq \{\{A \sim A\} \triangleright \{B \sim C\}$
create	$\text{cr}\langle A \rangle \triangleq \emptyset \triangleright \{A \sim A\}$
wait	$\text{wait}\langle r \rangle \triangleq r \triangleright r$
fail	$\text{fail}\langle r \rangle \triangleq r \triangleright \emptyset$

## Basic actions

swap	$\text{sw}\langle A \sim B @ C \rangle \triangleq \{\{A \sim C, B \sim C\} \triangleright \{A \sim B\}$
transmit	$\text{tr}\langle A \rightarrow B \sim C \rangle \triangleq \{\{A \sim A\} \triangleright \{B \sim C\}$
create	$\text{cr}\langle A \rangle \triangleq \emptyset \triangleright \{A \sim A\}$
wait	$\text{wait}\langle r \rangle \triangleq r \triangleright r$
fail	$\text{fail}\langle r \rangle \triangleq r \triangleright \emptyset$

## Probabilistic basic actions

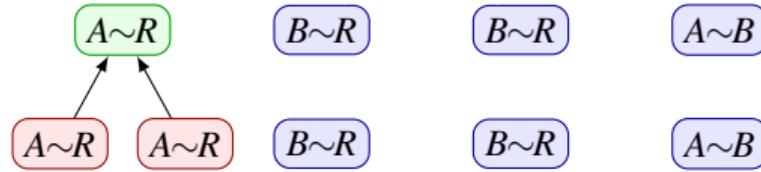
$$r \triangleright o + r \triangleright \emptyset \triangleq r \triangleright o + \text{fail}\langle r \rangle$$

# Probabilistic basic actions

Example (Distillation is inherently probabilistic.)

$$\text{di}\langle A \sim R \rangle \triangleq \{\{A \sim R, A \sim R\} \triangleright \{A \sim R\} + \{A \sim R, A \sim R\} \triangleright \emptyset\}$$

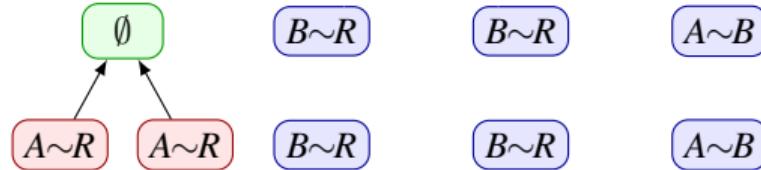
succeed :



input a :

fail :

input a :

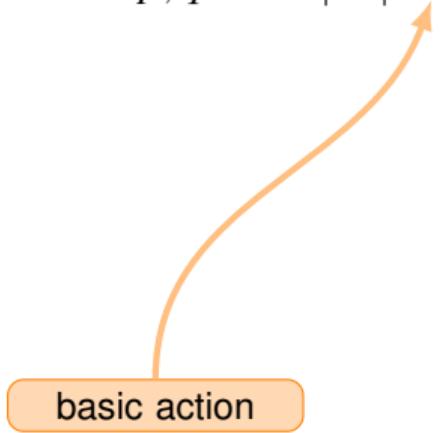


Bell pairs: **consumed**, **produced** and **untouched**

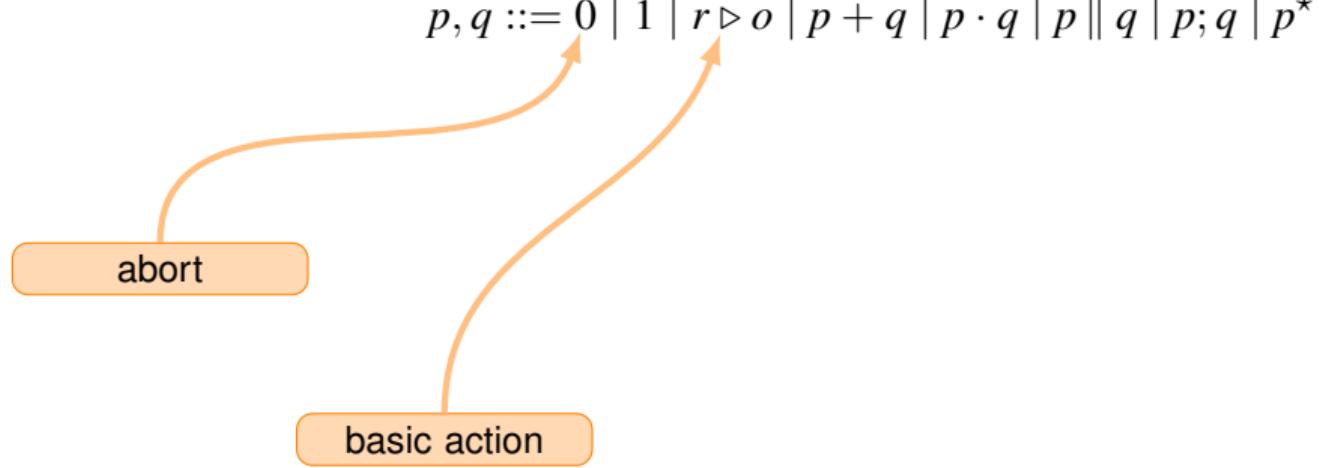
# Protocols

$$p, q ::= 0 \mid 1 \mid r \triangleright o \mid p + q \mid p \cdot q \mid p \parallel q \mid p; q \mid p^*$$

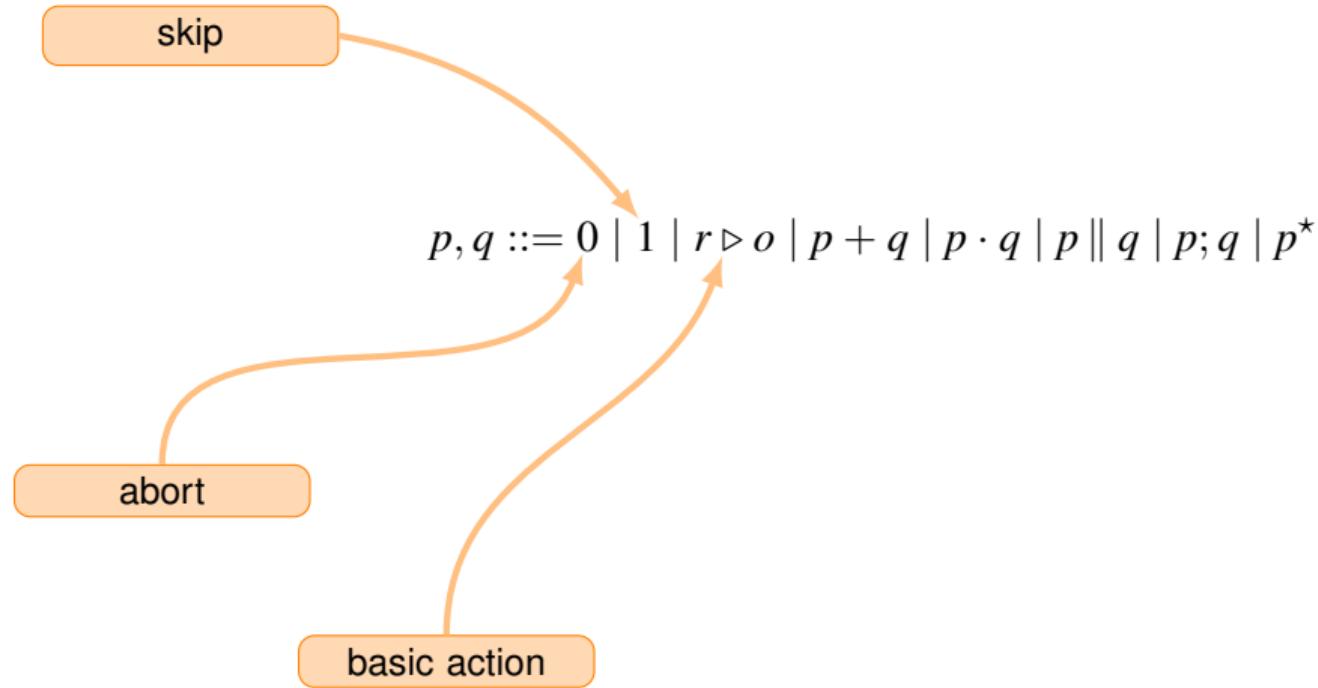
# Protocols

$$p, q ::= 0 \mid 1 \mid r \triangleright o \mid p + q \mid p \cdot q \mid p \parallel q \mid p; q \mid p^*$$


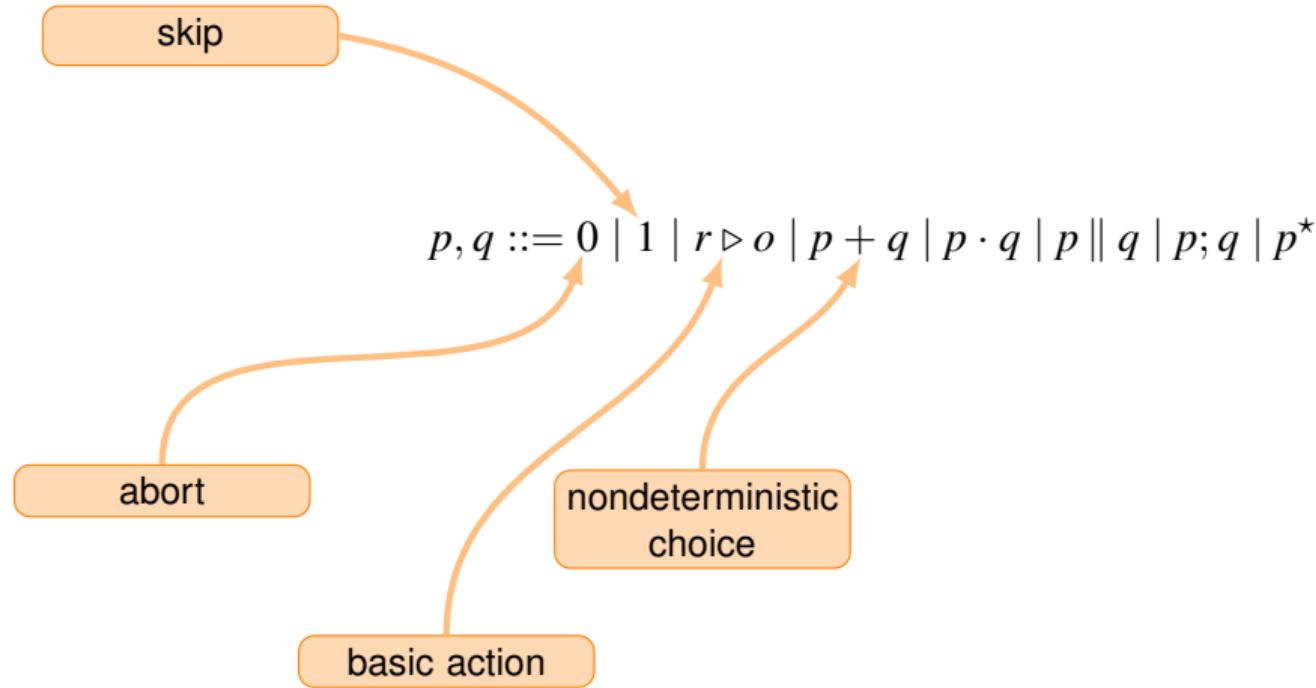
# Protocols

$$p, q ::= 0 \mid 1 \mid r \triangleright o \mid p + q \mid p \cdot q \mid p \parallel q \mid p; q \mid p^*$$


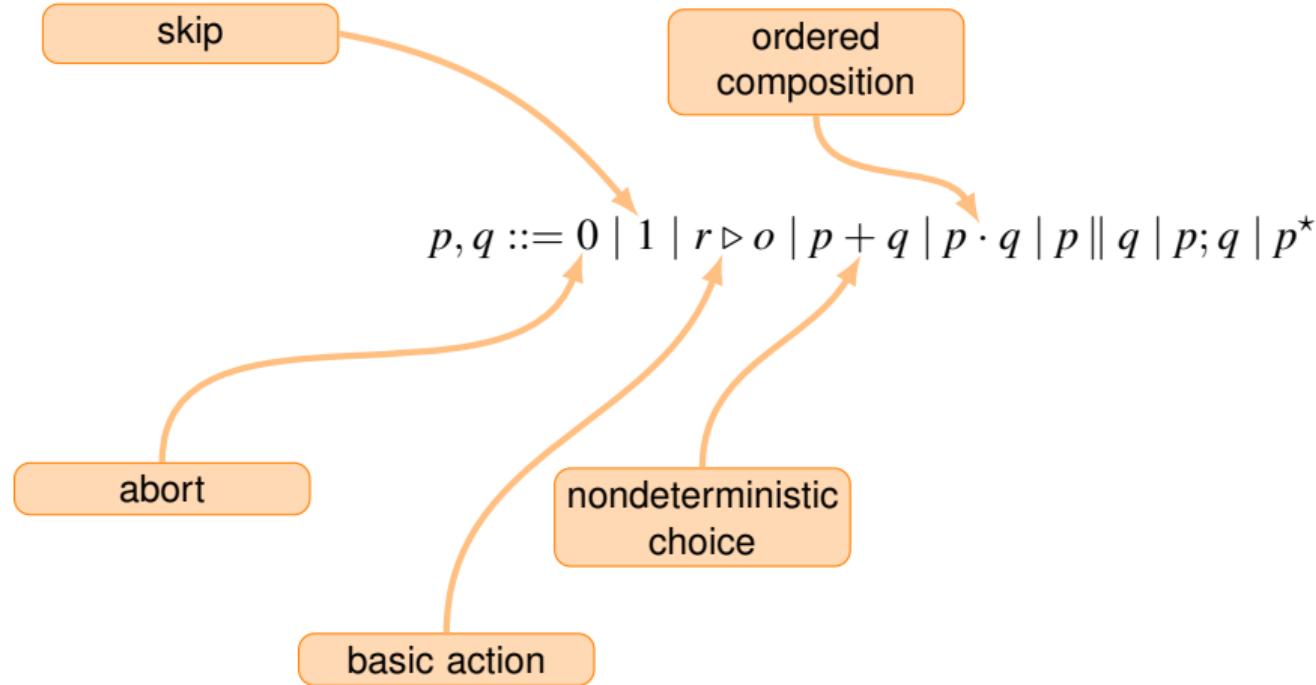
# Protocols



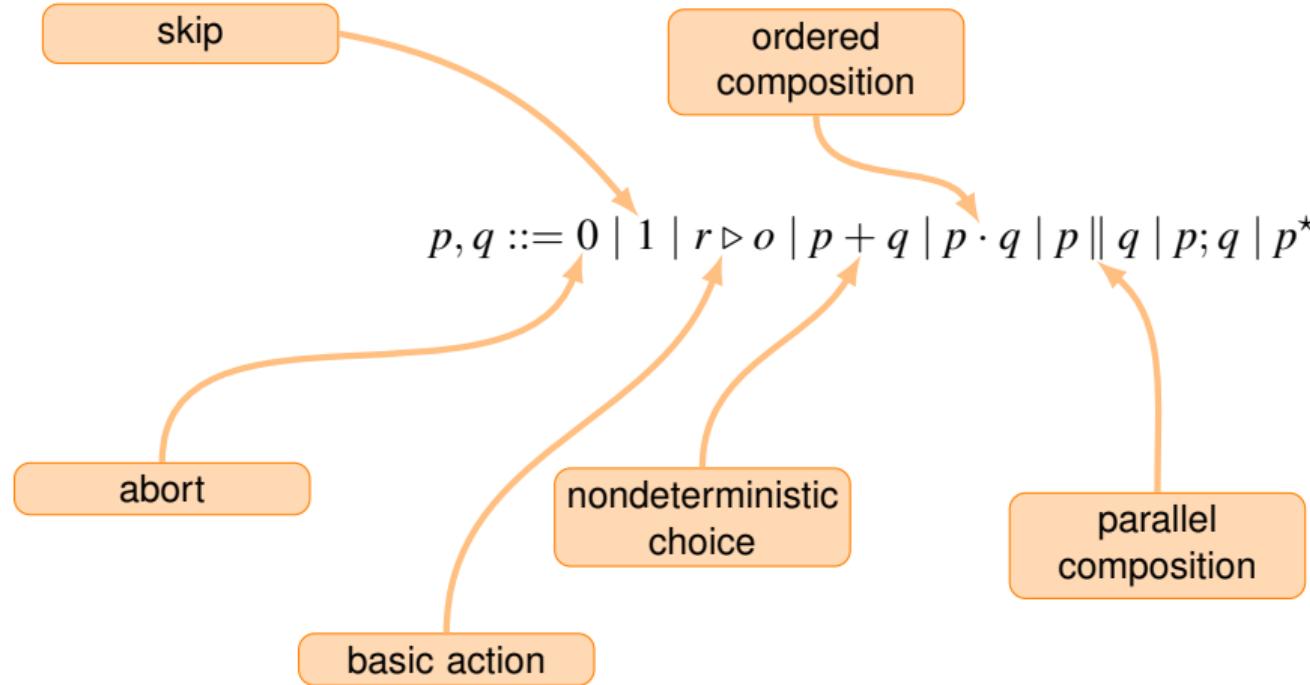
# Protocols



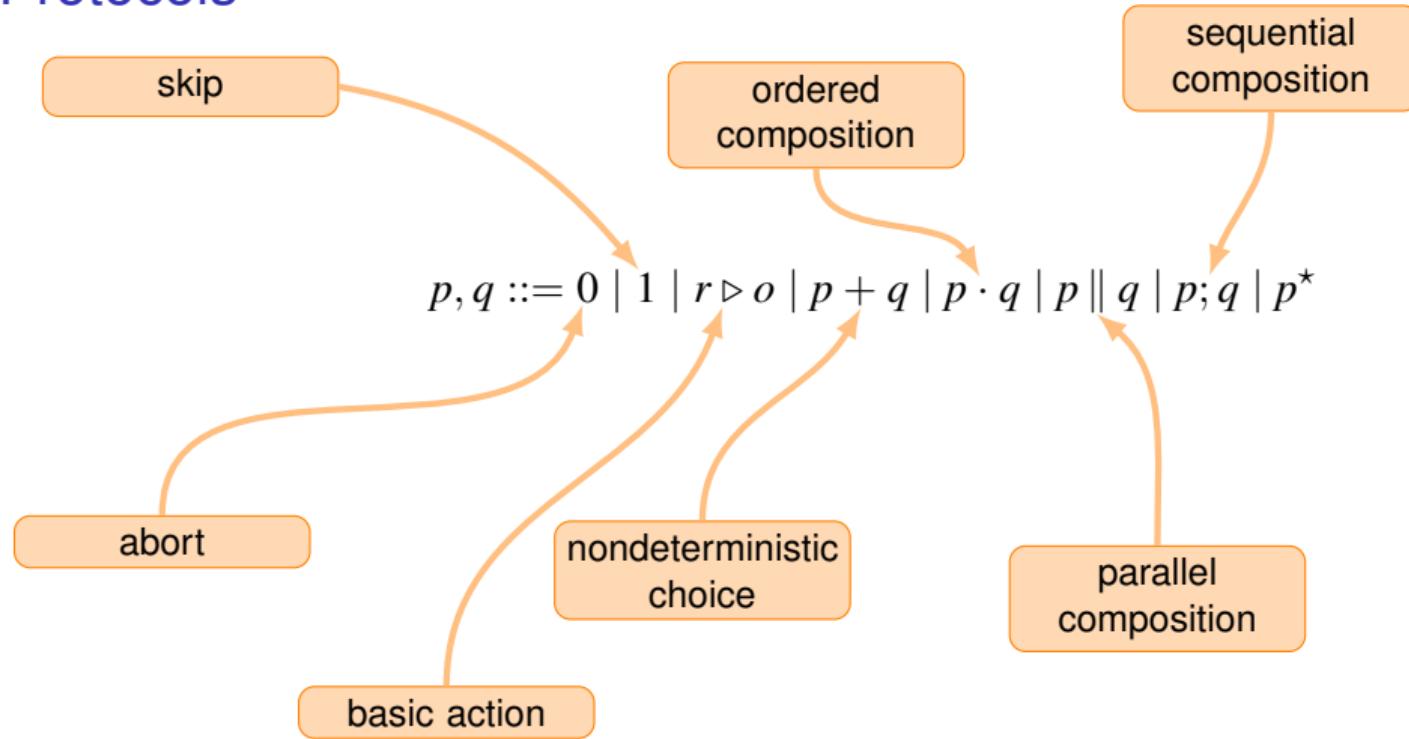
# Protocols



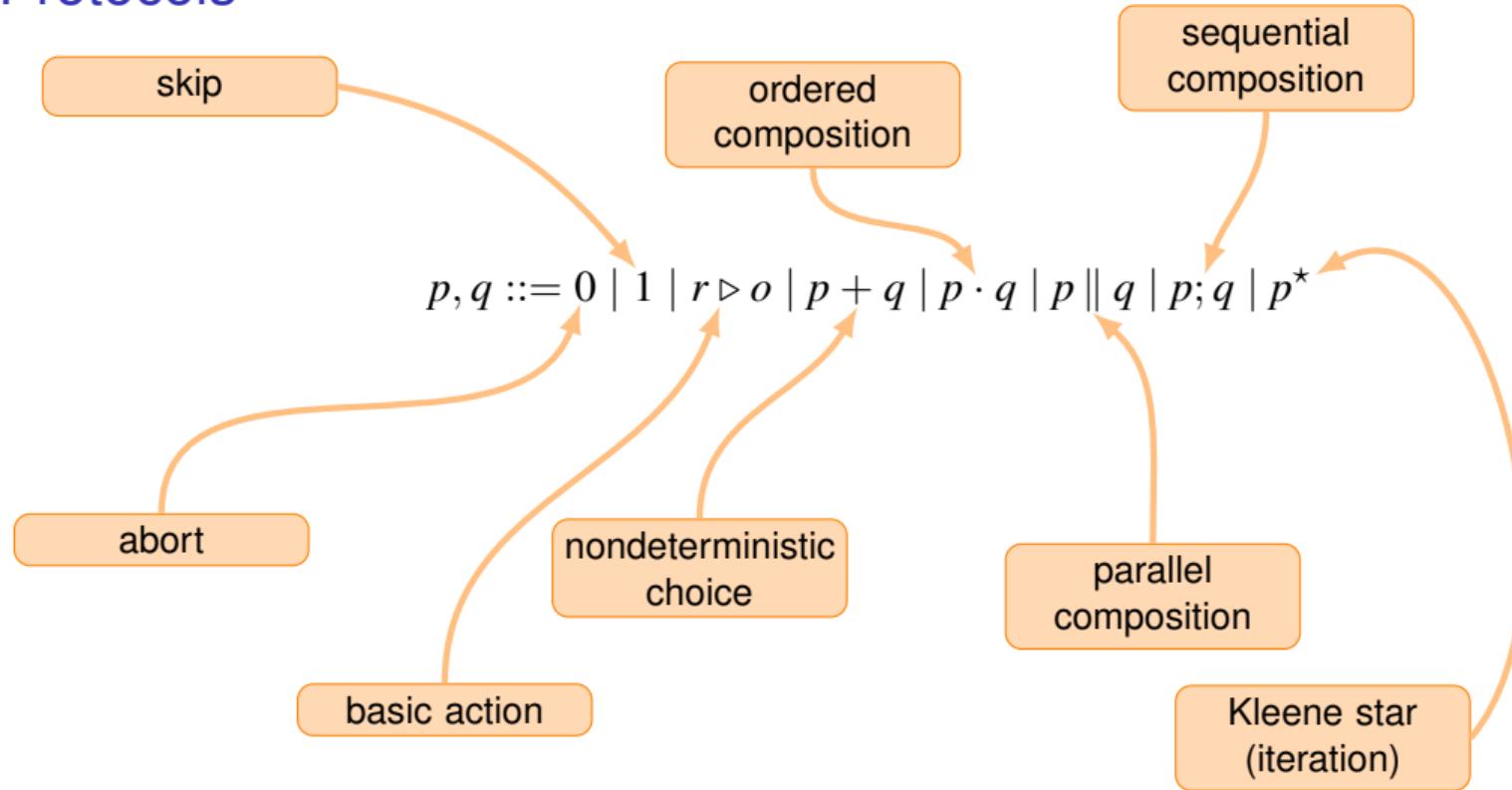
# Protocols



# Protocols



# Protocols



# Single round protocols

## Parallel composition

$\text{sw}\langle A \sim B @ R \rangle \parallel \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \parallel \text{sw}\langle B \sim C @ R' \rangle$  acts on  $\{A \sim R, B \sim R, B \sim R, B \sim R', C \sim R', A \sim B\}$

$A \sim R$      $B \sim R$      $B \sim R$      $B \sim R'$      $C \sim R'$      $A \sim B$

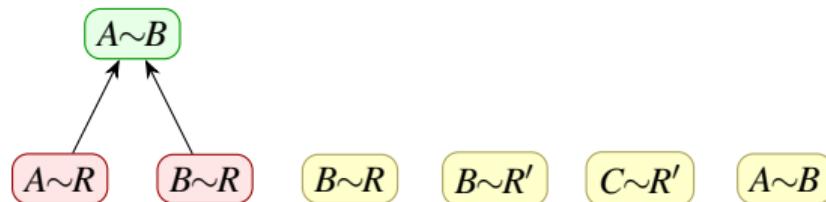
---

Input Bell pairs

# Single round protocols

## Parallel composition

$\text{sw}\langle A \sim B @ R \rangle \| \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \| \text{sw}\langle B \sim C @ R' \rangle$  acts on  $\{\underline{A \sim R}, \underline{B \sim R}, B \sim R, B \sim R', C \sim R', A \sim B\}$

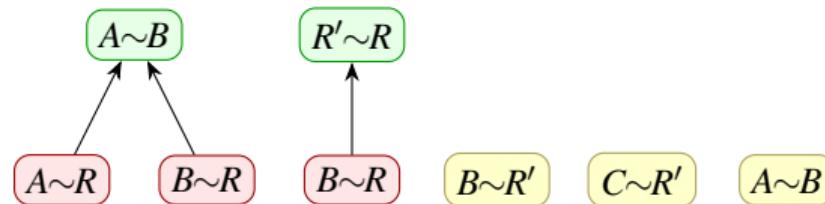


Bell pairs: **input** and **consumed**, **produced**

# Single round protocols

## Parallel composition

$\text{sw}\langle A \sim B @ R \rangle \parallel \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \parallel \text{sw}\langle B \sim C @ R' \rangle$  acts on  $\{A \sim R, B \sim R, B \sim R, B \sim R', C \sim R', A \sim B\}$

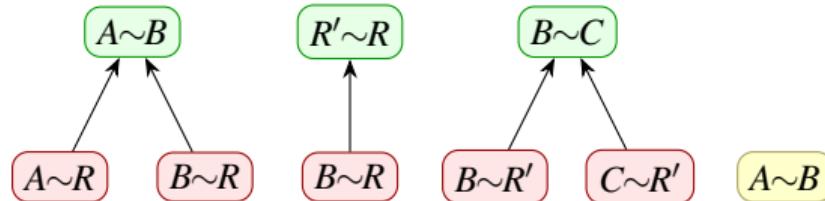


Bell pairs: **input** and **consumed**, **produced**

# Single round protocols

## Parallel composition

$\text{sw}\langle A \sim B @ R \rangle \parallel \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \parallel \text{sw}\langle B \sim C @ R' \rangle$  acts on  $\{A \sim R, B \sim R, B \sim R, \underline{B \sim R'}, C \sim R', A \sim B\}$

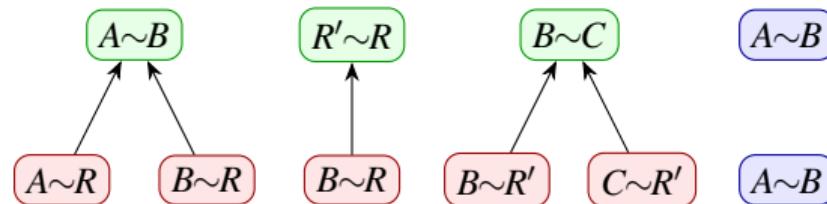


Bell pairs: input and consumed, produced

# Single round protocols

## Parallel composition

$\text{sw}\langle A \sim B @ R \rangle \parallel \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \parallel \text{sw}\langle B \sim C @ R' \rangle$  acts on  $\{A \sim R, B \sim R, B \sim R, B \sim R', C \sim R', A \sim B\}$

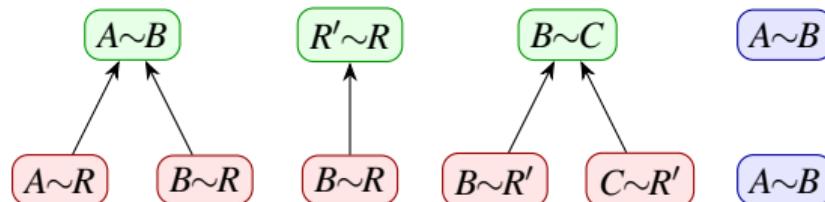


Bell pairs: **consumed**, **produced** and **untouched**

# Single round protocols

## Parallel composition

$\text{sw}\langle A \sim B @ R \rangle \parallel \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \parallel \text{sw}\langle B \sim C @ R' \rangle$  acts on  $\{A \sim R, B \sim R, B \sim R, B \sim R', C \sim R', A \sim B\}$



The order of basic actions is independent for this input multiset, thus:

$$\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle = \text{sw}\langle A \sim B @ R \rangle \parallel \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \parallel \text{sw}\langle B \sim C @ R' \rangle$$

---

Bell pairs: **consumed**, **produced** and **untouched**

# Parallel composition vs. ordered composition

$\mathbf{sw}\langle A \sim B @ R \rangle \| \mathbf{tr}\langle B \sim R \rightarrow R' \sim R \rangle \| \mathbf{sw}\langle B \sim C @ R' \rangle$

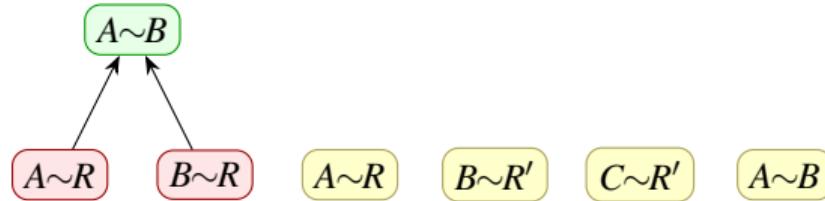
$A \sim R$      $B \sim R$      $A \sim R$      $B \sim R'$      $C \sim R'$      $A \sim B$

---

Bell pairs: **input** and **consumed**, **produced**

# Parallel composition vs. ordered composition

$\text{sw} \langle A \sim B @ R \rangle \cdot \text{tr} \langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw} \langle B \sim C @ R' \rangle$

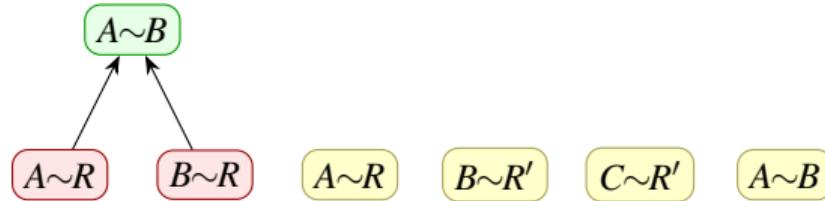


---

Bell pairs: **input** and **consumed**, **produced**

# Parallel composition vs. ordered composition

$\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$

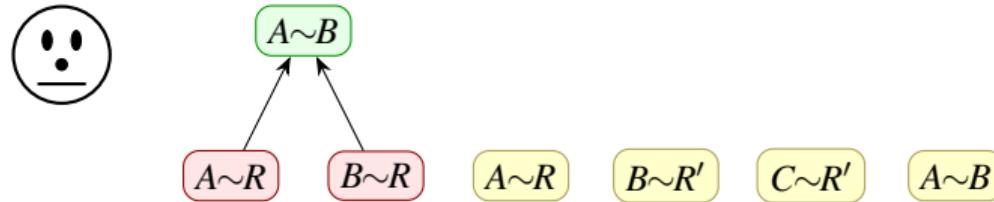


---

Bell pairs: **input** and **consumed**, **produced**

# Parallel composition vs. ordered composition

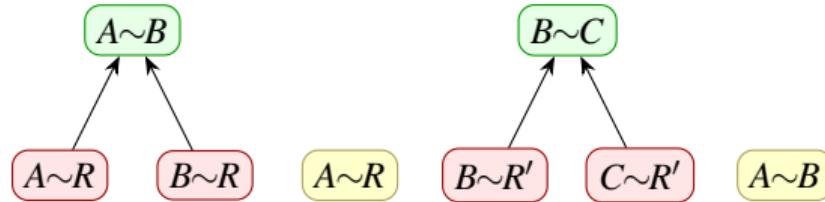
$$\text{sw} \langle A \sim B @ R \rangle \cdot \text{tr} \langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw} \langle B \sim C @ R' \rangle$$



Bell pairs: input and consumed, produced

# Parallel composition vs. ordered composition

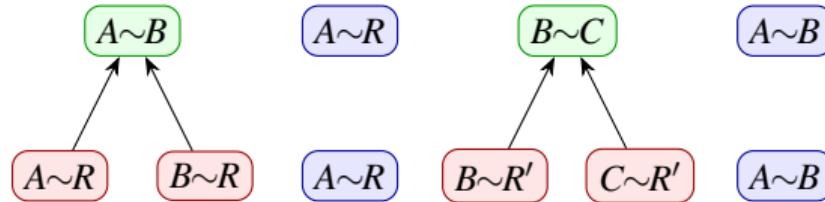
$$\text{sw} \langle A \sim B @ R \rangle \cdot \text{tr} \langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw} \langle B \sim C @ R' \rangle$$



<sup>2</sup>Bell pairs: input and consumed, produced.

# Parallel composition vs. ordered composition

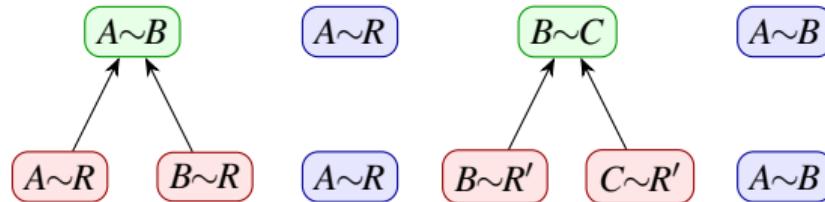
$$\text{sw} \langle A \sim B @ R \rangle \cdot \text{tr} \langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw} \langle B \sim C @ R' \rangle$$



Bell pairs: **consumed**, **produced** and **untouched**

# Parallel composition vs. ordered composition

$\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$



$\text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle A \sim B @ R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$

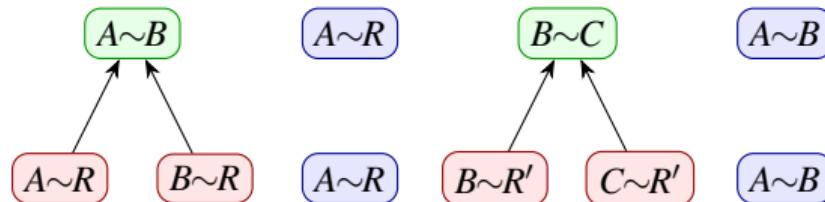


---

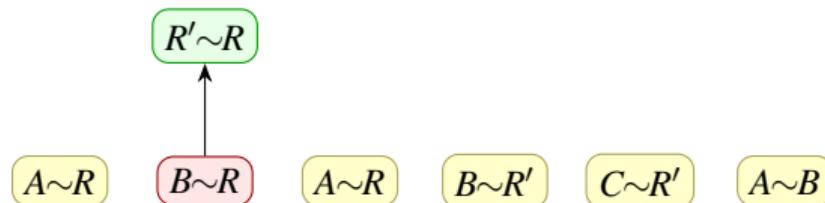
Bell pairs: **consumed**, **produced** and **untouched**

# Parallel composition vs. ordered composition

$$\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$$



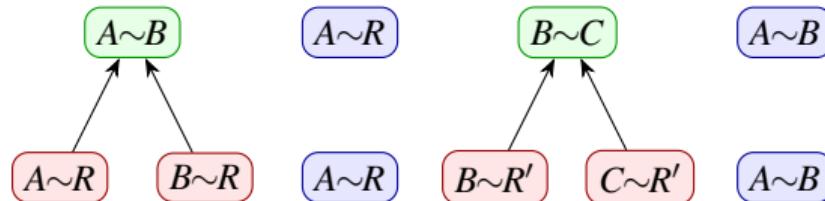
$$\text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle A \sim B @ R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$$



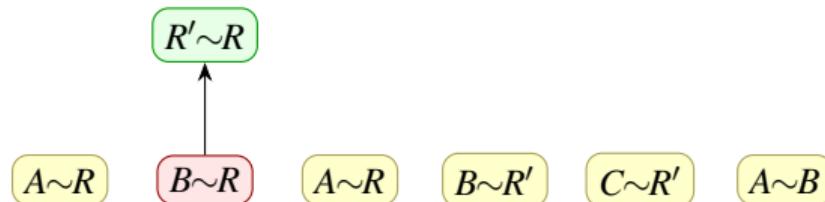
Bell pairs: **consumed**, **produced** and **untouched**

# Parallel composition vs. ordered composition

$\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$



$\text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle A \sim B @ R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$

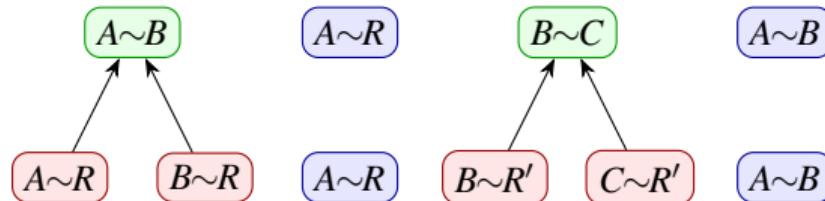


---

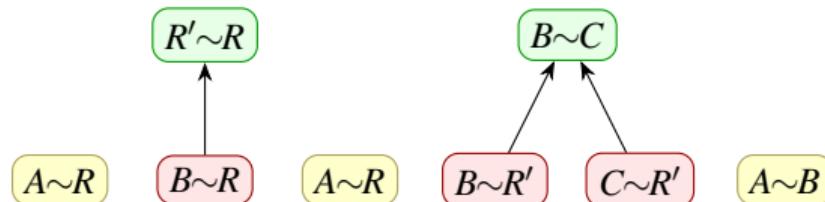
Bell pairs: **consumed**, **produced** and **untouched**

# Parallel composition vs. ordered composition

$$\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$$



$$\text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle A \sim B @ R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$$

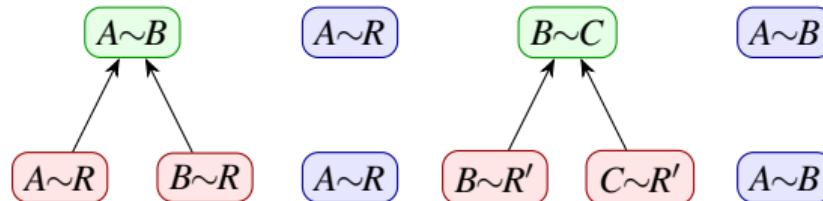


---

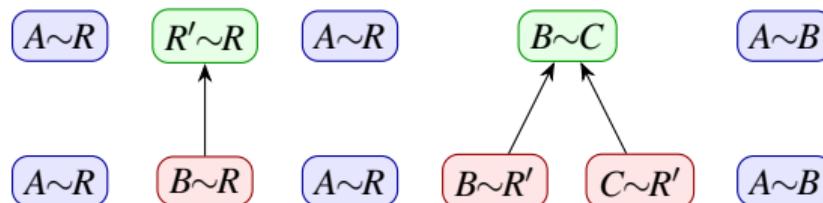
Bell pairs: **consumed**, **produced** and **untouched**

# Parallel composition vs. ordered composition

$$\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$$



$$\text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle A \sim B @ R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$$

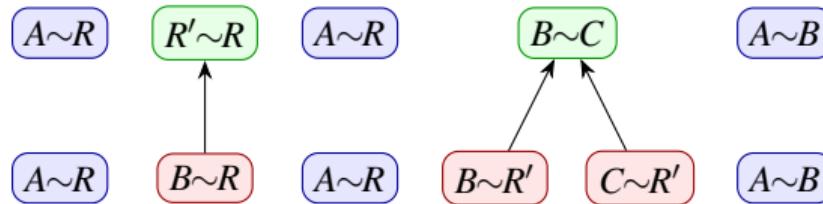
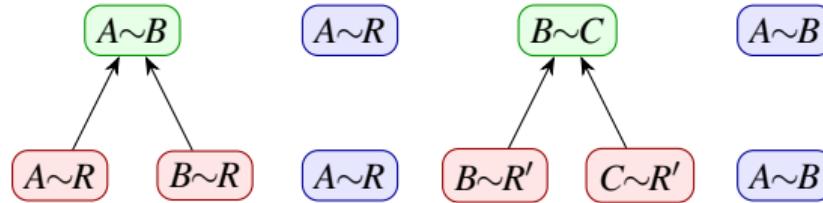


---

Bell pairs: **consumed**, **produced** and **untouched**

# Parallel composition vs. ordered composition

$\text{sw}\langle A \sim B @ R \rangle \| \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \| \text{sw}\langle B \sim C @ R' \rangle$



---

Bell pairs: **consumed**, **produced** and **untouched**

## Verification - properties with a classical analogue

- *Reachability.* Will the execution of our protocol generate  $A \sim C$ ?

## Verification - properties with a classical analogue

- *Reachability.* Will the execution of our protocol generate  $A \sim C$ ?
- *Waypoint Correctness.* Does our protocol ever perform a swap at node  $B$ ?

## Verification - properties with a classical analogue

- *Reachability.* Will the execution of our protocol generate  $A \sim C$ ?
- *Waypoint Correctness.* Does our protocol ever perform a swap at node  $B$ ?
- *Traffic (Protocol) Isolation.* Can we prove non-interference in a composition of Protocols I and II?

## Verification - properties with a classical analogue

- *Reachability.* Will the execution of our protocol generate  $A \sim C$ ?
- *Waypoint Correctness.* Does our protocol ever perform a swap at node  $B$ ?
- *Traffic (Protocol) Isolation.* Can we prove non-interference in a composition of Protocols I and II?
- *Compilation.* Can we ensure correct compilation to individual devices?

## Verification - properties specific to quantum

- *Resource Utilization.* What is the number of required memory locations and communication qubits? For how many rounds must a Bell pair wait in the memory?

## Verification - properties specific to quantum

- *Resource Utilization.* What is the number of required memory locations and communication qubits? For how many rounds must a Bell pair wait in the memory?
- *Quality of Service.* Do the generated Bell pairs have the required fidelity or capacity?

## Verification - properties specific to quantum

- *Resource Utilization.* What is the number of required memory locations and communication qubits? For how many rounds must a Bell pair wait in the memory?
- *Quality of Service.* Do the generated Bell pairs have the required fidelity or capacity?
- *Compilation.* Can we minimize the number of accesses to the network global state?

# Verification - properties specific to quantum

- *Resource Utilization.* What is the number of required memory locations and communication qubits? For how many rounds must a Bell pair wait in the memory?
- *Quality of Service.* Do the generated Bell pairs have the required fidelity or capacity?
- *Compilation.* Can we minimize the number of accesses to the network global state?

Thank you!

