Towards an Algebraic Specification of Quantum Networks

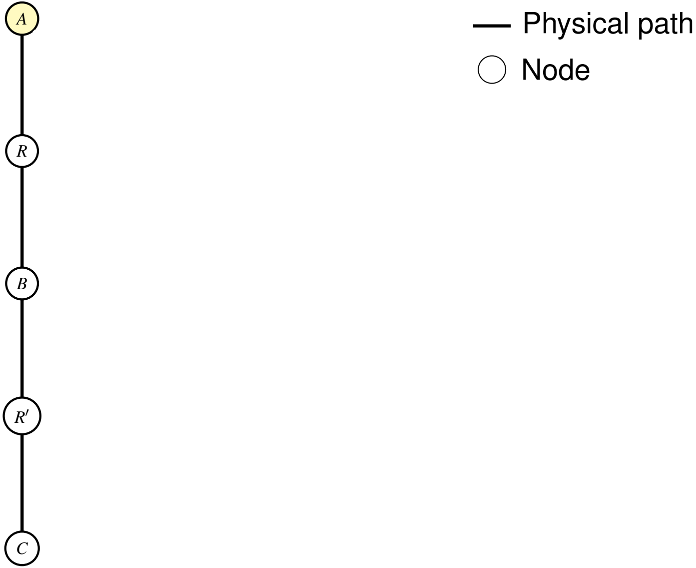
Anita Buckley[1], Pavel Chuprikov[1], Rodrigo Otoni[1],
Robert Rand[2], Robert Soulé[3] and Patrick Eugster[1]

[1] Università della Svizzera italiana (USI), Switzerland
[2] University of Chicago, USA
[3] Yale University, USA

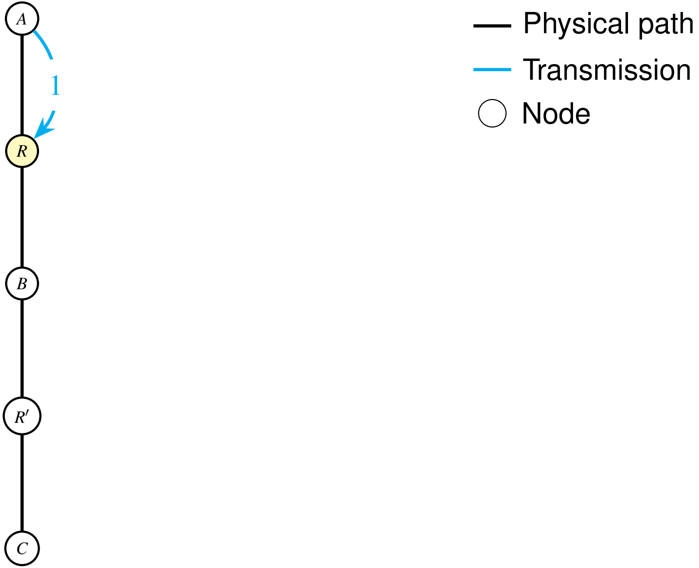
New York City, 10th September 2023

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

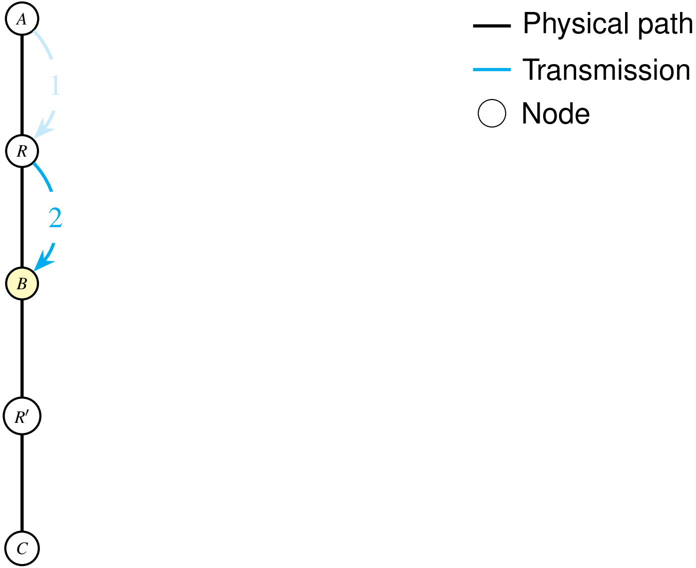
Classical packet forwarding



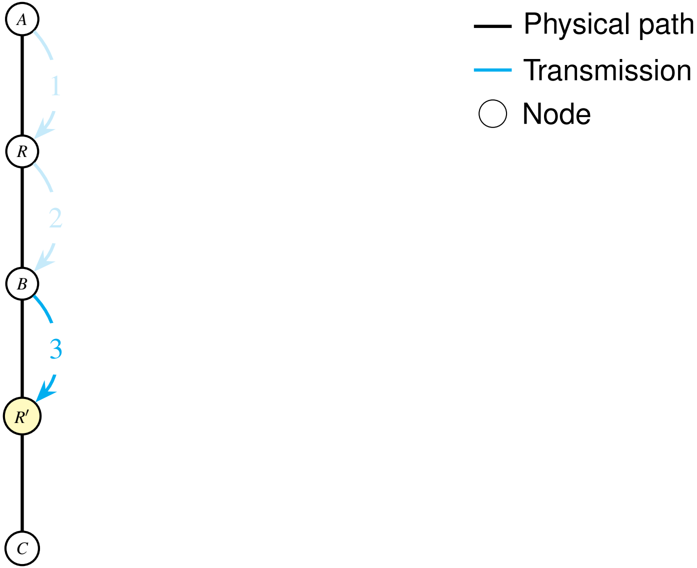
Classical packet forwarding



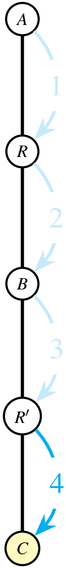
Classical packet forwarding

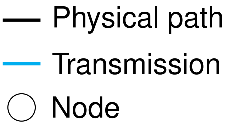


Classical packet forwarding



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 has 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 the design of NetKAT, including primitives for filtering, modifying, and transmitting packets; union and sequential composition operators; and a Kleene star operator that iterates programs. We show that NetKAT is an instance of a canonical and well-studied 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 reachability, 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, Frenetic, 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 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 from hosts, topology changes, and shifts in traffic load by re-programming the switches accordingly. Because the controller 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 major appeal 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, thereby making it easier to build sophisticated and reliable SDN applications. For example, the languages developed in the Frenetic 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 expressed by having the general-purpose program generate a sequence of static policies—and simplicity—forwarding policies are written in a simple domain-specific language with a clear 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 Frenetic [6] used simple lists of predicate-action rules as policies, where the actions in-

* This work performed at Cornell University.

POPL '14, January 22–24, 2014, San Diego, CA, USA.

http://dx.doi.org/10.1145/2535838.2535862

NetKAT: Semantic Foundations for Networks

Carolyn Jane Anderson, Swarthmore College *
Nate Foster, Cornell University
Arjun Guha
Jean-Baptiste Jeannin, Carnegie Mellon University *
Dexter Kozen, Cornell University
Cole Schles…, Princeton Un…

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 has 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 the design of NetKAT, including primitives for filtering, modifying, and transmitting packets; union and sequential composition operators; and a Kleene star operator that iterates programs. We show that NetKAT is an instance of a canonical and well-studied 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 reachability, 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, Frenetic, 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.

POPL '14, January 22–24, 2014, San Diego, CA, USA.
Copyright © 2014 ACM 978-1-4503-2544-8/14/01…$15.00.
http://dx.doi.org/10.1145/2535838.2535862

Probabilistic NetKAT

Nate Foster[1], Dexter Kozen[1], Konstantinos Mamouras[2*], Mark Reitblatt[3*], and Alexandra Silva[4]

[1] Cornell University
[2] University of Pennsylvania
[3] Facebook
[4] 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. We establish fundamental properties of the semantics, prove that it is a conservative extension of the deterministic semantics, show that it satisfies a number of natural equations, and develop a notion of approximation. We present case studies that show how the language can be used to model a diverse collection of scenarios drawn from real-world networks.

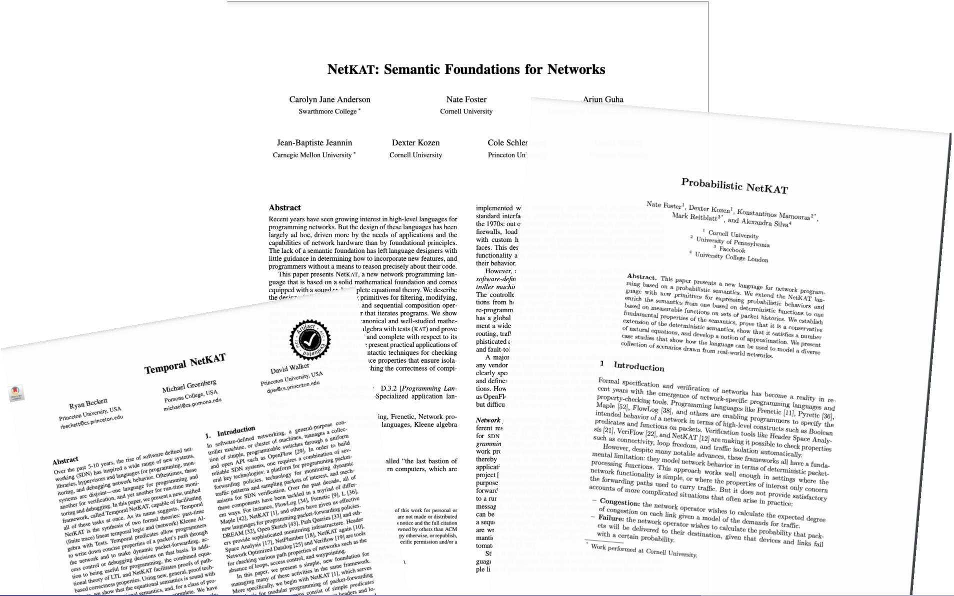
1 Introduction

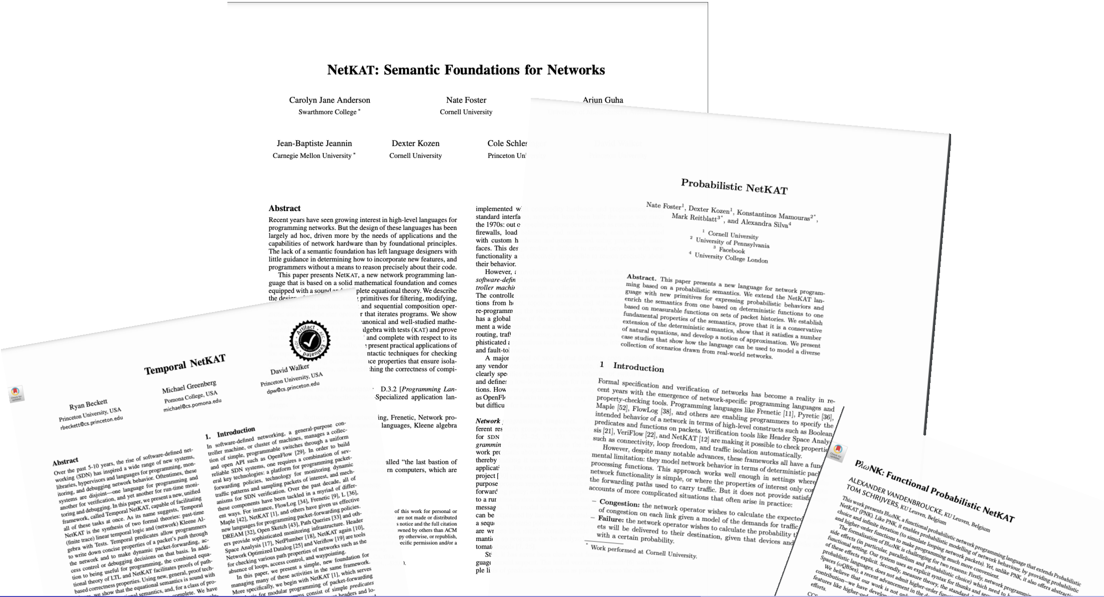
Formal specification and verification of networks has become a reality in recent years with the emergence of network-specific programming languages and property-checking tools. Programming languages like Frenetic [11], Pyretic [36], Maple [52], FlowLog [38], and others are enabling programmers to specify 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 connectivity, 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 functionality 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.
– **Failure:** 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.

* Work performed at Cornell University.


NetKAT: Semantic Foundations for Networks
Carolyn Jane Anderson
Swarthmore College *
Nate Foster
Cornell University
Arjun Guha
Jean-Baptiste Jeannin
Carnegie Mellon University *
Dexter Kozen
Cornell 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 has left language designers with little guidance in determining how to incorporate new features, and programmers without a means to reason precisely about their code.
Probabilistic NetKAT
Nate Foster[1], Dexter Kozen[1], Konstantinos Mamouras[2*], Mark Reitblatt[3*], and Alexandra Silva[4]
[1] Cornell University
[2] University of Pennsylvania
[3] Facebook
[4] 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. We establish fundamental properties of the semantics, prove that it is a conservative extension of the deterministic semantics, show that it satisfies a number of natural equations, and develop a notion of approximation. 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
Temporal NetKAT
Ryan Beckett
Princeton University, USA
rbeckett@cs.princeton.edu
Michael Greenberg
Pomona College, USA
michael@cs.pomona.edu
David Walker
Princeton University, USA
dpw@cs.princeton.edu
Abstract
1. Introduction



NetKAT: Semantic Foundations for Networks

Carolyn Jane Anderson, Swarthmore College*
Nate Foster, Cornell University
Arjun Guha
Jean-Baptiste Jeannin, Carnegie Mellon University*
Dexter Kozen, Cornell University
Cole Schles[...], Princeton Uni[...]

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 has left language designers with little guidance in determining how to incorporate new features, and programmers without a means to reason precisely about their code.

Probabilistic NetKAT

Nate Foster[1], Dexter Kozen[1], Konstantinos Mamouras[2*], Mark Reitblatt[3*], and Alexandra Silva[4]
[1] Cornell University
[2] University of Pennsylvania
[3] Facebook
[4] 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. We establish fundamental properties of the semantics, prove that it is a conservative extension of the deterministic semantics, show that it satisfies a number of natural equations, and develop a notion of approximation. We present case studies that show how the language can be used to model a diverse collection of scenarios drawn from real-world networks.

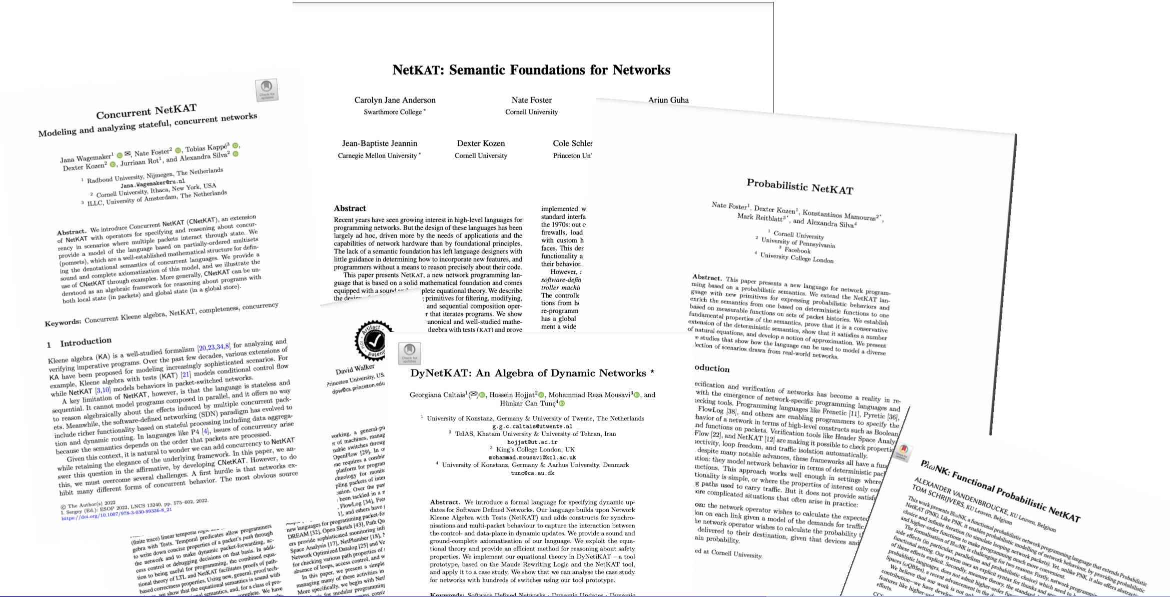
Temporal NetKAT

Ryan Beckett, Princeton University, USA, rbeckett@cs.princeton.edu
Michael Greenberg, Pomona College, USA, michael@cs.pomona.edu
David Walker, Princeton University, USA, dpw@cs.princeton.edu

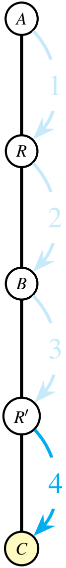
P$\lambda\omega$NK: Functional Probabilistic NetKAT

ALEXANDER VANDENBROUCKE, KU Leuven, Belgium
TOM SCHRIJVERS, KU Leuven, Belgium


NetKAT: Semantic Foundations for Networks
Carolyn Jane Anderson, Swarthmore College
Nate Foster, Cornell University
Arjun Guha
Jean-Baptiste Jeannin, Carnegie Mellon University
Dexter Kozen, Cornell 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 has 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
Probabilistic NetKAT
Nate Foster[1], Dexter Kozen[1], Konstantinos Mamouras[2], Mark Reitblatt[3], and Alexandra Silva[4]
[1] Cornell University
[2] University of Pennsylvania
[3] Facebook
[4] 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. We establish fundamental properties of the semantics, prove that it is a conservative extension of the deterministic semantics, show that it satisfies a number of natural equations, and develop a notion of approximation. We present case studies that show how the language can be used to model a diverse collection of scenarios drawn from real-world networks.
Temporal NetKAT
Ryan Beckett, Princeton University, USA, rbeckett@cs.princeton.edu
Michael Greenberg, Pomona College, USA, michael@cs.pomona.edu
David Walker, Princeton University, USA, dpw@cs.princeton.edu
Abstract
1. Introduction
DyNetKAT: An Algebra of Dynamic Networks *
Georgiana Caltais[1], Hossein Hojjat[2], Mohammad Reza Mousavi[3], and Hünkar Can Tunç[4]
[1] University of Konstanz, Germany & University of Twente, The Netherlands g.g.c.caltais@utwente.nl
[2] TeIAS, Khatam University & University of Tehran, Iran hojjat@ut.ac.ir
[3] King's College London, UK mohammad.mousavi@kcl.ac.uk
[4] University of Konstanz, Germany & Aarhus University, Denmark tunc@cs.au.dk
Abstract. We introduce a formal language for specifying dynamic updates for Software Defined Networks. Our language builds upon Network Kleene Algebra with Tests (NetKAT) and adds constructs for synchronisations and multi-packet behaviour to capture the interaction between the control- and data-plane in dynamic updates. We provide a sound and ground-complete axiomatisation of our language. We exploit the equational theory and provide an efficient method for reasoning about safety properties. We implement our equational theory in DyNetiKAT – a tool prototype, based on the Maude Rewriting Logic and the NetKAT tool, and apply it to a case study. We show that we can analyse the case study for networks with hundreds of switches using our tool prototype.
ProbNetKAT: Functional Probabilistic NetKAT
ALEXANDER VANDENBROUCKE, KU Leuven, Belgium
TOM SCHRIJVERS, KU Leuven, Belgium


Concurrent NetKAT
Modeling and analyzing stateful, concurrent networks
Jana Wagemaker, Nate Foster, Tobias Kappé, Dexter Kozen, Jurriaan Rot, and Alexandra Silva
NetKAT: Semantic Foundations for Networks
Carolyn Jane Anderson
Swarthmore College
Nate Foster
Cornell University
Jean-Baptiste Jeannin
Carnegie Mellon University
Dexter Kozen
Cornell University
Abstract
Probabilistic NetKAT
Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva
DyNetKAT: An Algebra of Dynamic Networks
Georgiana Caltais, Hossein Hojjat, Mohammad Reza Mousavi, and Hünkar Can Tunç
PλωNK: Functional Probabilistic NetKAT
ALEXANDER VANDENBROUCKE, KU Leuven, Belgium
TOM SCHRIJVERS, KU Leuven, Belgium

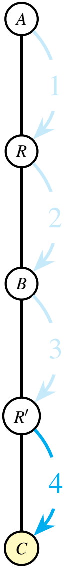
NetKAT expression $p^\star$ is encoding network's behavior



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

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

NetKAT expression $p^\star$ is encoding network's behavior

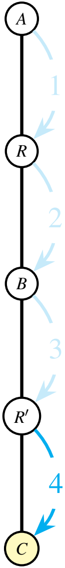


test

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

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

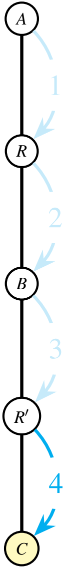
NetKAT expression $p^\star$ is encoding network's behavior



$$\begin{aligned} p \equiv\ & (\underbrace{\mathsf{sw} = A}_{\text{test}}; \underbrace{\mathsf{sw} \leftarrow R}_{\text{action}}) + (\mathsf{sw} = R; \mathsf{sw} \leftarrow B) + \\ & (\mathsf{sw} = B; \mathsf{sw} \leftarrow R') + (\mathsf{sw} = R'; \mathsf{sw} \leftarrow C) \end{aligned}$$

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

NetKAT expression $p^\star$ is encoding network's behavior



action

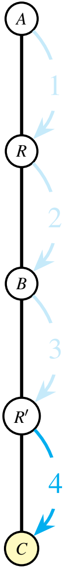
test

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

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

sequential composition

NetKAT expression $p^\star$ is encoding network's behavior



test

action

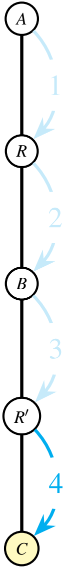
$$p \equiv (\mathsf{sw} = A; \mathsf{sw} \leftarrow R) + (\mathsf{sw} = R; \mathsf{sw} \leftarrow B) + (\mathsf{sw} = B; \mathsf{sw} \leftarrow R') + (\mathsf{sw} = R'; \mathsf{sw} \leftarrow C)$$

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

nondeterministic choice

sequential composition

NetKAT expression $p^\star$ is encoding network's behavior



test

action

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

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

iteration

nondeterministic choice

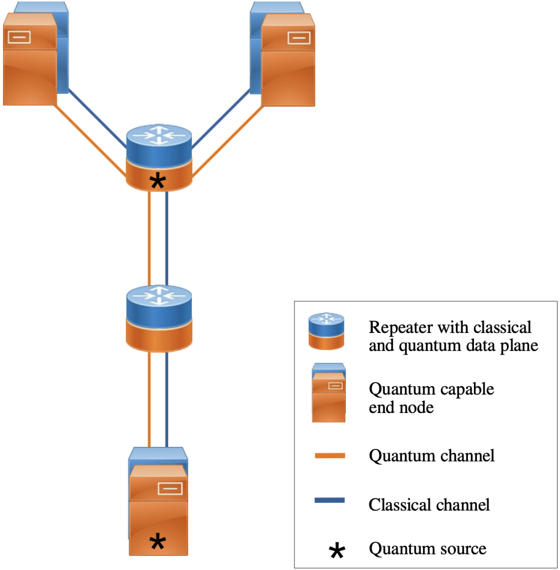
sequential composition

Artwork by Sandbox Studio, Chicago with Ana Kova

’spooky action at a distance’

Bell pair: maximally entangled quantum bits

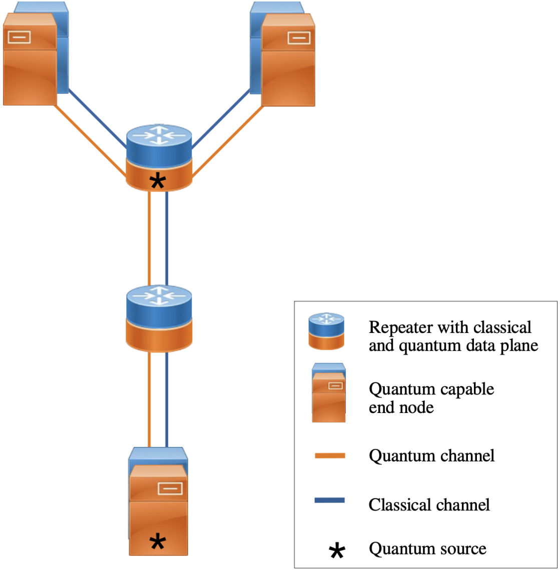
- fundamental unit in quantum networks
- consists of two quantum bits (qubits): $R{\sim}B$ is a Bell pair distributed between nodes R and B
- no headers: control information needs to be sent via separate classical channels



1

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

[1] W. Kozlowski and S. Wehner: *Towards Large-Scale Quantum Networks.* NANOCOM (2019)
[2] IRTF, QIRG: *Architectural Principles for a Quantum Internet.* RFC 9340 (2023)



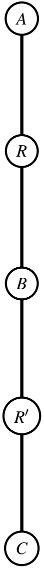
1

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

end-to-end Bell pair distribution[2]

[1] W. Kozlowski and S. Wehner: *Towards Large-Scale Quantum Networks.* NANOCOM (2019)
[2] IRTF, QIRG: *Architectural Principles for a Quantum Internet.* RFC 9340 (2023)

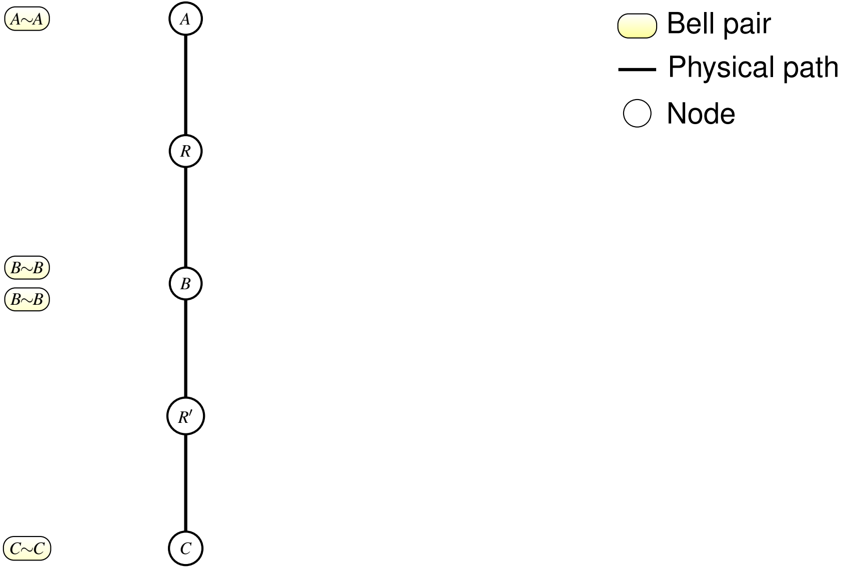
Bell pair generation



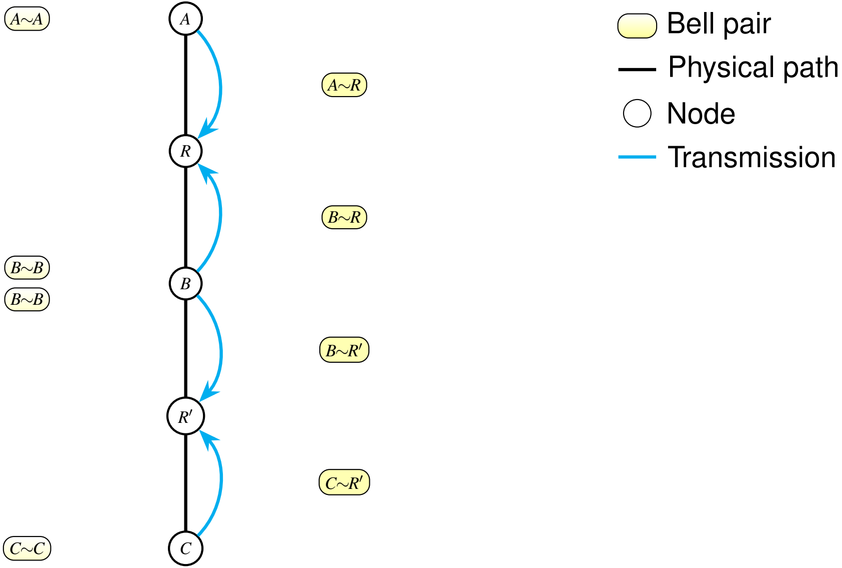
— Physical path

◯ Node

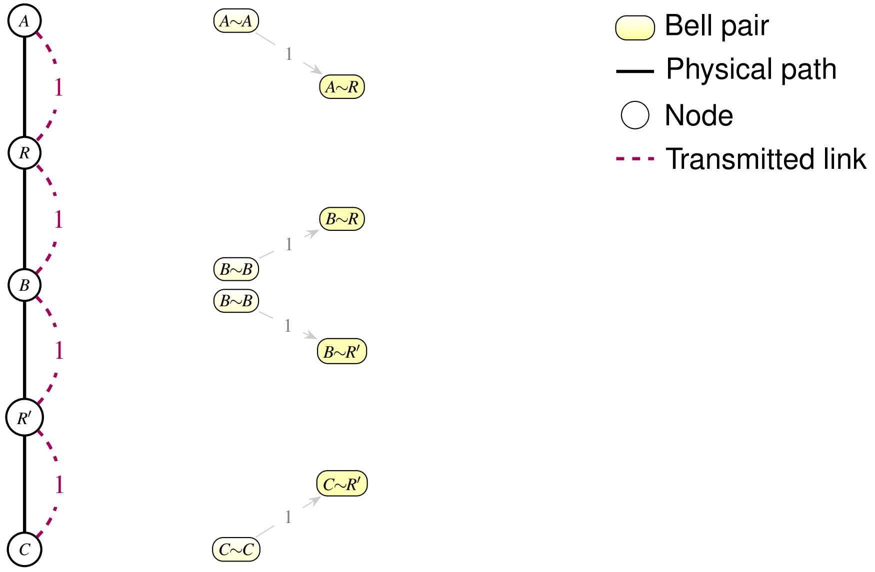
Bell pair generation: Protocol I



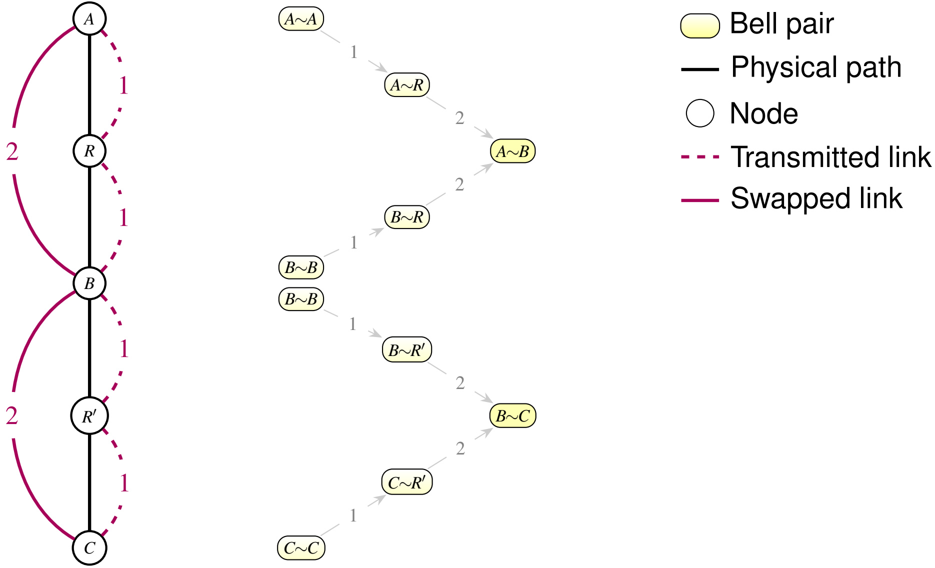
Bell pair generation: Protocol I



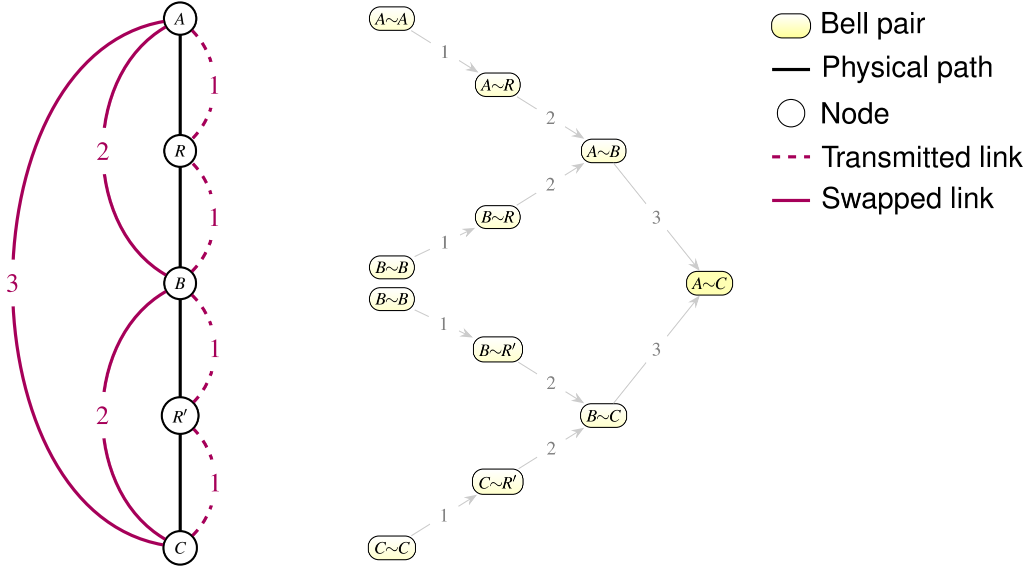
Bell pair generation: Protocol I, round 1



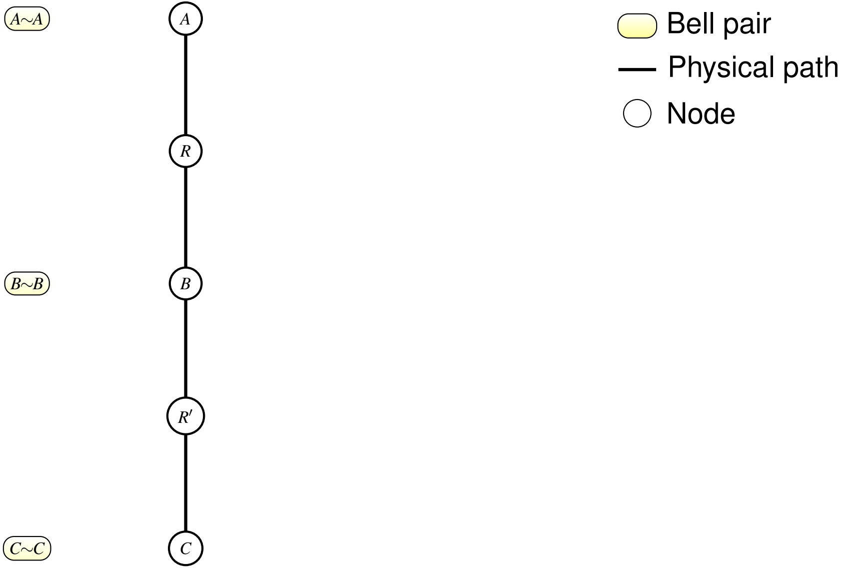
Bell pair generation: Protocol I, round 2



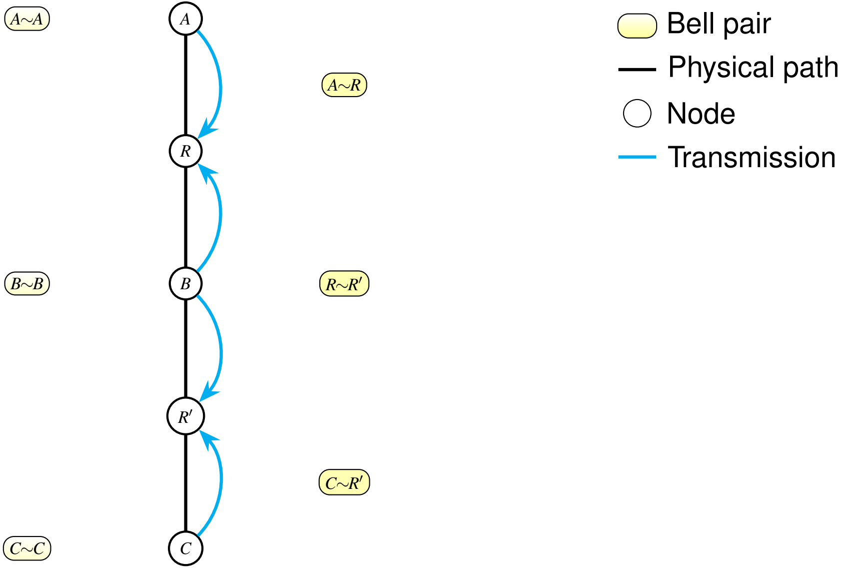
Bell pair generation: Protocol I, round 3



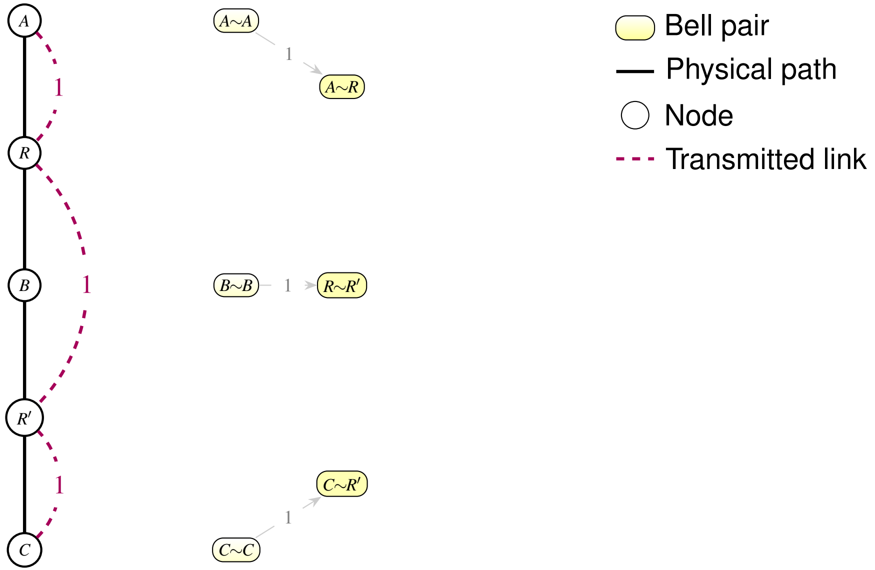
Bell pair generation: Protocol II



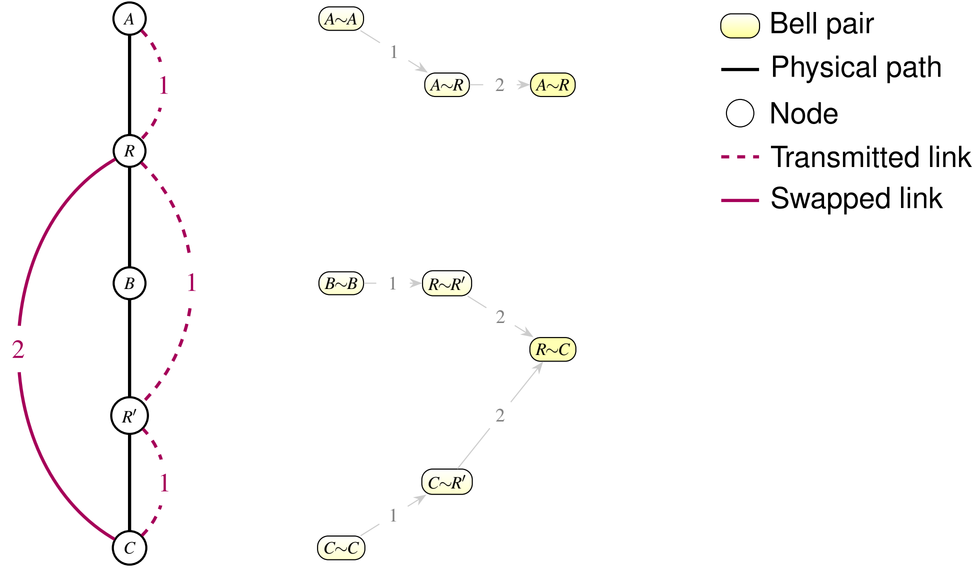
Bell pair generation: Protocol II



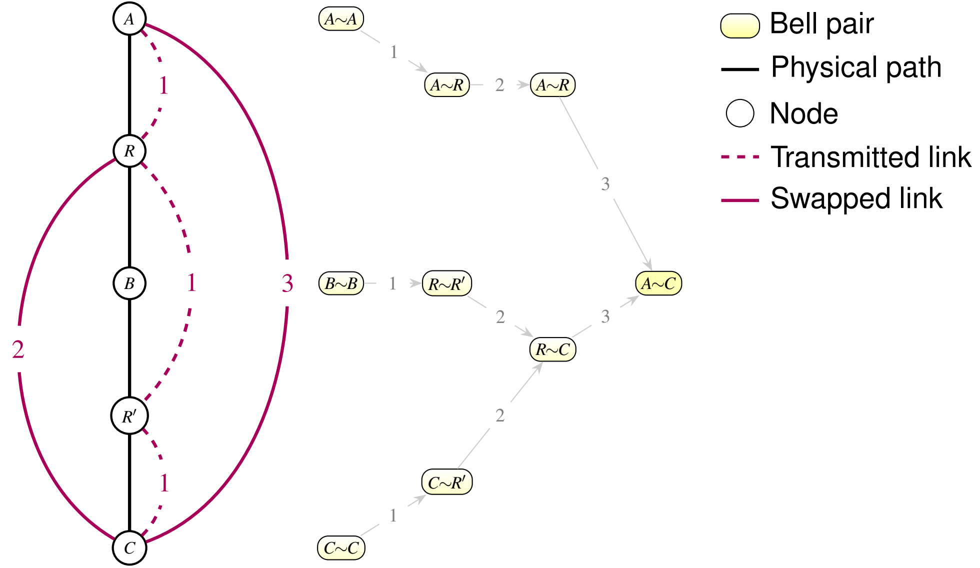
Bell pair generation: Protocol II, round 1



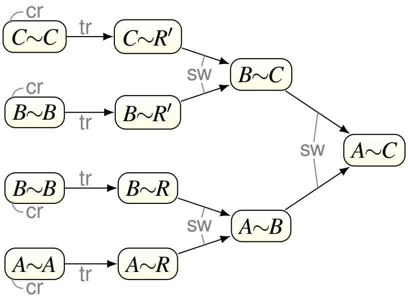
Bell pair generation: Protocol II, round 2

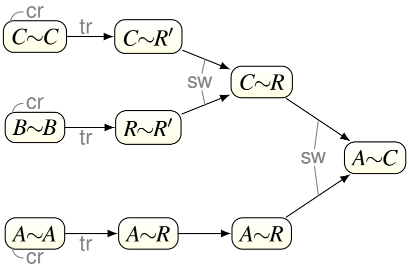


Bell pair generation: Protocol II, round 3

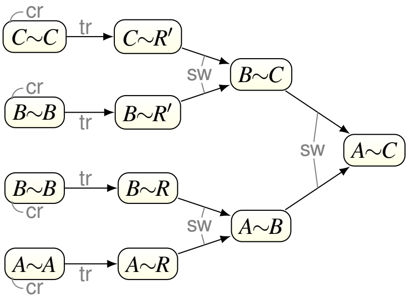


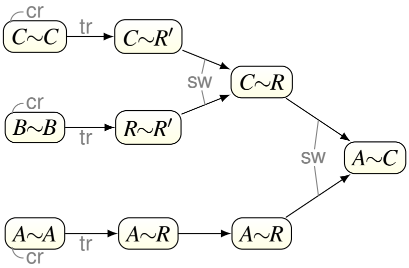
Specification: Protocol I and Protocol II generating $A{\sim}C$





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





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

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

Basic action $r \triangleright o$

$$
\begin{aligned}
r \triangleright o \colon \mathcal{M}(\mathsf{BP}) \to &\quad \mathcal{M}(\mathsf{BP}) \times \mathcal{M}(\mathsf{BP}) \\
a \quad \mapsto & \begin{cases} o \bowtie a \backslash r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}
\end{aligned}
$$

$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$

required
Bell pairs

$$
\begin{aligned}
r \triangleright o \colon \mathcal{M}(\mathsf{BP}) &\to \quad \mathcal{M}(\mathsf{BP}) \times \mathcal{M}(\mathsf{BP}) \\
a \quad &\mapsto \begin{cases} o \bowtie a \backslash r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}
\end{aligned}
$$

$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$

required Bell pairs

output Bell pairs

$$r \triangleright o \colon \mathcal{M}(\mathsf{BP}) \to \mathcal{M}(\mathsf{BP}) \times \mathcal{M}(\mathsf{BP})$$

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

$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$

required Bell pairs

output Bell pairs

input multiset of Bell pairs

$$r \triangleright o \colon \mathcal{M}(\mathsf{BP}) \to \mathcal{M}(\mathsf{BP}) \times \mathcal{M}(\mathsf{BP})$$

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

$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$

required Bell pairs

output Bell pairs

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

input multiset of Bell pairs

produced Bell pairs

$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$

required Bell pairs

output Bell pairs

$$r \triangleright o \colon \mathcal{M}(\mathsf{BP}) \to \quad \mathcal{M}(\mathsf{BP}) \times \mathcal{M}(\mathsf{BP})$$

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

input multiset of Bell pairs

produced Bell pairs

consumed Bell pairs

$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$

required Bell pairs

output Bell pairs

$$r \triangleright o \colon \mathcal{M}(\mathsf{BP}) \to \mathcal{M}(\mathsf{BP}) \times \mathcal{M}(\mathsf{BP})$$

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

input multiset of Bell pairs

produced Bell pairs

untouched Bell pairs

consumed Bell pairs

$\bowtie$: pair of multisets of Bell pairs

Basic action

$$\begin{aligned} r \triangleright o \colon \mathcal{M}(\mathsf{BP}) &\to \quad \mathcal{M}(\mathsf{BP}) \times \mathcal{M}(\mathsf{BP}) \\ a \quad &\mapsto \begin{cases} o \bowtie a \backslash r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases} \end{aligned}$$

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

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

$A{\sim}R$ $B{\sim}R$ $A{\sim}R$ $B{\sim}R$ $A{\sim}B$

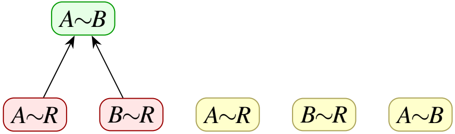
Input Bell pairs

Basic action

$$
\begin{aligned}
r \triangleright o \colon \mathcal{M}(\mathsf{BP}) &\to \quad \mathcal{M}(\mathsf{BP}) \times \mathcal{M}(\mathsf{BP}) \\
a \quad &\mapsto \begin{cases} o \bowtie a \backslash r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}
\end{aligned}
$$

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

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



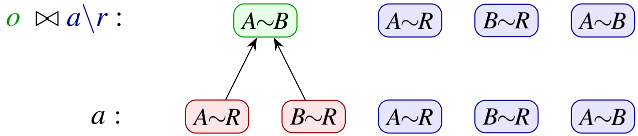
Bell pairs: input and consumed, produced

Basic action

$$
\begin{aligned}
r \triangleright o \colon \mathcal{M}(\mathrm{BP}) &\to \quad \mathcal{M}(\mathrm{BP}) \times \mathcal{M}(\mathrm{BP}) \\
a \quad &\mapsto \begin{cases} o \bowtie a \backslash r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}
\end{aligned}
$$

Swap action 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: consumed, produced and untouched

Basic action

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

Swap action 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

$$
\begin{aligned}
r \triangleright o \colon \mathcal{M}(\mathsf{BP}) &\to \quad \mathcal{M}(\mathsf{BP}) \times \mathcal{M}(\mathsf{BP}) \\
a \quad &\mapsto \begin{cases} o \bowtie a \backslash r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}
\end{aligned}
$$

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

$\{\!\{A{\sim}R, B{\sim}R\}\!\} \triangleright \{\!\{A{\sim}B\}\!\}$ 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

$$r \triangleright o \colon \mathcal{M}(\mathrm{BP}) \to \mathcal{M}(\mathrm{BP}) \times \mathcal{M}(\mathrm{BP})$$
$$a \mapsto \begin{cases} o \bowtie a \backslash r & \text{if } r \subseteq a \\ \emptyset \bowtie a & \text{otherwise} \end{cases}$$

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

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

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

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

Bell pairs: consumed, produced and untouched

Basic actions

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

IRTF, QIRG: *Architectural Principles for a Quantum Internet.* RFC 9340 (2023)

Basic actions

$$\begin{array}{cr} \text{swap} & \mathsf{sw}\langle A{\sim}B \mathbin{@} C\rangle \triangleq \{\!\{A{\sim}C, B{\sim}C\}\!\} \triangleright \{\!\{A{\sim}B\}\!\} \\ \text{transmit} & \mathsf{tr}\langle A \to B{\sim}C\rangle \triangleq \{\!\{A{\sim}A\}\!\} \triangleright \{\!\{B{\sim}C\}\!\} \\ \text{create} & \mathsf{cr}\langle A\rangle \triangleq \emptyset \triangleright \{\!\{A{\sim}A\}\!\} \\ \text{wait} & \mathsf{wait}\langle r\rangle \triangleq r \triangleright r \\ \text{fail} & \mathsf{fail}\langle r\rangle \triangleq r \triangleright \emptyset \end{array}$$

Probabilistic basic actions

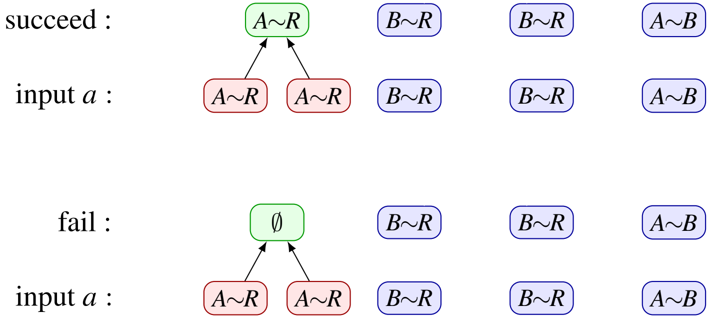
$$r \triangleright o + r \triangleright \emptyset \triangleq r \triangleright o + \mathsf{fail}\langle r\rangle$$

IRTF, QIRG: *Architectural Principles for a Quantum Internet.* RFC 9340 (2023)

Probabilistic basic actions

Example (Distillation is inherently probabilistic.)

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



Bell pairs: consumed, produced and untouched

Protocols

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

Protocols

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

basic action

Protocols

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

abort

basic action

Protocols

skip

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

abort

basic action

Protocols

skip

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

abort

nondeterministic choice

basic action

Protocols

skip

ordered composition

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

abort

nondeterministic choice

basic action

Protocols

skip

ordered composition

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

abort

basic action

nondeterministic choice

parallel composition

Protocols

skip

ordered composition

sequential composition

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

abort

nondeterministic choice

parallel composition

basic action

Protocols

skip

ordered composition

sequential composition

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

abort

nondeterministic choice

parallel composition

basic action

Kleene star (iteration)

Single round protocols

Parallel composition

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

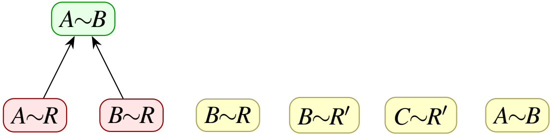
$A{\sim}R$ $B{\sim}R$ $B{\sim}R$ $B{\sim}R'$ $C{\sim}R'$ $A{\sim}B$

Input Bell pairs

Single round protocols

Parallel composition

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

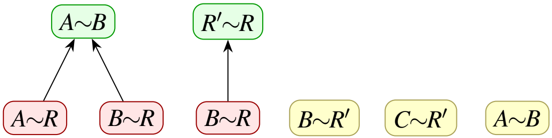


Bell pairs: input and consumed, produced

Single round protocols

Parallel composition

$\underline{\mathsf{sw}\langle A{\sim}B \,@\, R\rangle \parallel \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle} \parallel \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle$ acts on $\{\!\{\cancel{A{\sim}R}, \cancel{B{\sim}R}, \underline{B{\sim}R}, B{\sim}R', C{\sim}R', A{\sim}B\}\!\}$

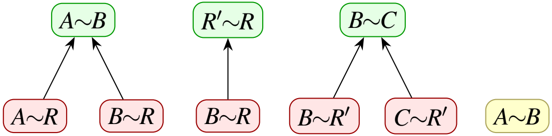


Bell pairs: input and consumed, produced

Single round protocols

Parallel composition

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

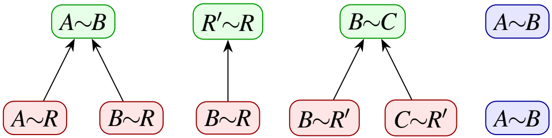


Bell pairs: input and consumed, produced

Single round protocols

Parallel composition

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

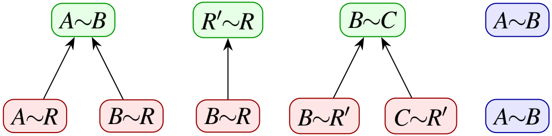


Bell pairs: consumed, produced and untouched

Single round protocols

Parallel composition

$\mathsf{sw}\langle A{\sim}B \mathbin{@} R\rangle \parallel \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \parallel \mathsf{sw}\langle B{\sim}C \mathbin{@} R'\rangle$ 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:

$$\mathsf{sw}\langle A{\sim}B \mathbin{@} R\rangle \cdot \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \cdot \mathsf{sw}\langle B{\sim}C \mathbin{@} R'\rangle = \mathsf{sw}\langle A{\sim}B \mathbin{@} R\rangle \parallel \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \parallel \mathsf{sw}\langle B{\sim}C \mathbin{@} R'\rangle$$

Bell pairs: consumed, produced and untouched

Parallel composition vs. ordered composition

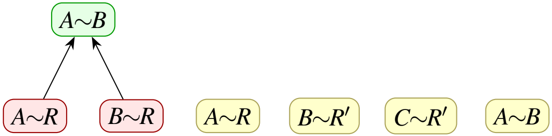
$\mathsf{sw}\langle A{\sim}B \,@\, R\rangle \parallel \mathsf{tr}\langle B{\sim}R \rightarrow R'{\sim}R\rangle \parallel \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle$

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

Bell pairs: input and consumed, produced

Parallel composition vs. ordered composition

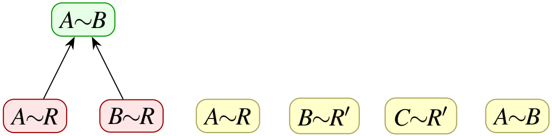
$\underline{\mathsf{sw}\langle A{\sim}B \mathbin{@} R\rangle} \cdot \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \cdot \mathsf{sw}\langle B{\sim}C \mathbin{@} R'\rangle$



Bell pairs: input and consumed, produced

Parallel composition vs. ordered composition

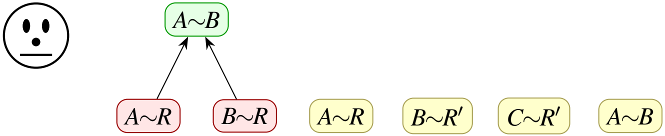
$$\underline{\mathsf{sw}\langle A{\sim}B \mathbin{@} R\rangle \cdot \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle} \cdot \mathsf{sw}\langle B{\sim}C \mathbin{@} R'\rangle$$



Bell pairs: input and consumed, produced

Parallel composition vs. ordered composition

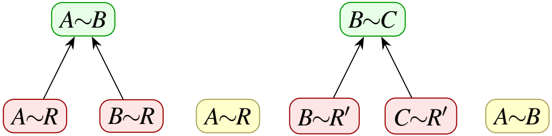
$$\underline{\mathsf{sw}\langle A{\sim}B \mathbin{@} R\rangle \cdot \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle} \cdot \mathsf{sw}\langle B{\sim}C \mathbin{@} R'\rangle$$



Bell pairs: input and consumed, produced

Parallel composition vs. ordered composition

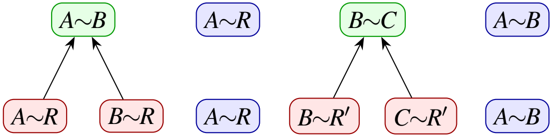
$$\underline{\mathsf{sw}\langle A{\sim}B \,@\, R\rangle \cdot \mathsf{tr}\langle B{\sim}R \rightarrow R'{\sim}R\rangle \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle}$$



[2] Bell pairs: input and consumed, produced.

Parallel composition vs. ordered composition

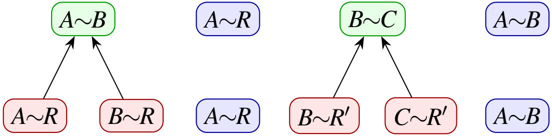
$\mathsf{sw}\langle A{\sim}B \,@\, R\rangle \cdot \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle$



Bell pairs: consumed, produced and untouched

Parallel composition vs. ordered composition

$\underline{\mathsf{sw}\langle A{\sim}B \,@\, R\rangle \cdot \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle}$

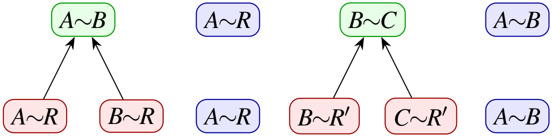


$\mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \cdot \mathsf{sw}\langle A{\sim}B \,@\, R\rangle \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle$

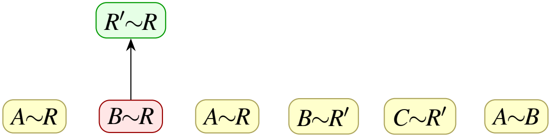
Bell pairs: consumed, produced and untouched

Parallel composition vs. ordered composition

$\underline{\mathsf{sw}\langle A{\sim}B \,@\, R\rangle \cdot \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle}$



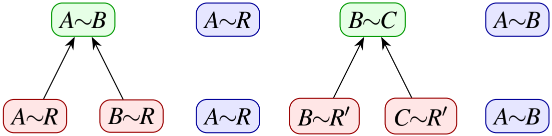
$\underline{\mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle} \cdot \mathsf{sw}\langle A{\sim}B \,@\, R\rangle \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle$



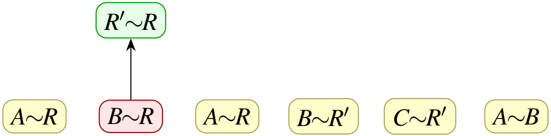
Bell pairs: consumed, produced and untouched

Parallel composition vs. ordered composition

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



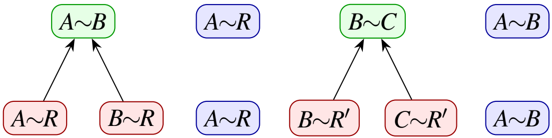
$\underline{\mathsf{tr}\langle B{\sim}R \rightarrow R'{\sim}R\rangle \cdot \mathsf{sw}\langle A{\sim}B \,@\, R\rangle} \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle$



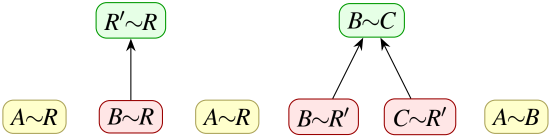
Bell pairs: consumed, produced and untouched

Parallel composition vs. ordered composition

$\underline{\mathsf{sw}\langle A{\sim}B \,@\, R\rangle \cdot \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle}$



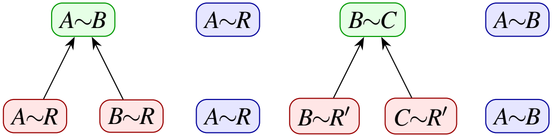
$\underline{\mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \cdot \mathsf{sw}\langle A{\sim}B \,@\, R\rangle \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle}$



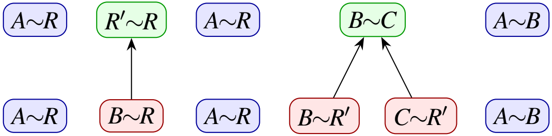
Bell pairs: consumed, produced and untouched

Parallel composition vs. ordered composition

$\underline{\mathsf{sw}\langle A{\sim}B \,@\, R\rangle \cdot \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle}$



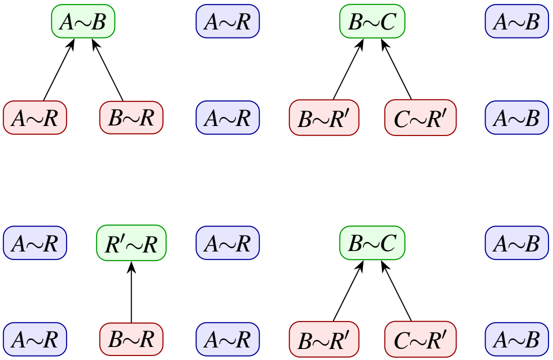
$\underline{\mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \cdot \mathsf{sw}\langle A{\sim}B \,@\, R\rangle \cdot \mathsf{sw}\langle B{\sim}C \,@\, R'\rangle}$



Bell pairs: consumed, produced and untouched

Parallel composition vs. ordered composition

$\mathsf{sw}\langle A{\sim}B @ R\rangle \parallel \mathsf{tr}\langle B{\sim}R \to R'{\sim}R\rangle \parallel \mathsf{sw}\langle B{\sim}C @ R'\rangle$



Bell pairs: consumed, produced and untouched

Verification - properties with a classical analogoue

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

Verification - properties with a classical analogoue

- *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 analogoue

- *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 analogoue

- *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!