

Towards an Algebraic Specification of Quantum Networks

Anita Buckley¹, Pavel Chuprikov¹, Rodrigo Otoni¹,
Robert Rand², Robert Soulé³ and Patrick Eugster¹

¹ Università della Svizzera italiana (USI), Switzerland

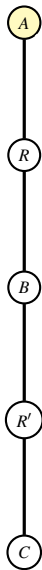
² University of Chicago, USA

³ Yale University, USA

New York City, 10th September 2023

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

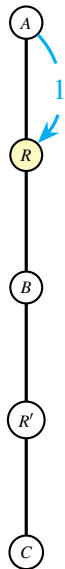
Classical packet forwarding



— Physical path

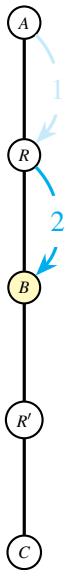
○ Node

Classical packet forwarding



- Physical path
- Transmission
- Node

Classical packet forwarding

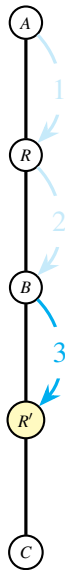


— Physical path

— Transmission

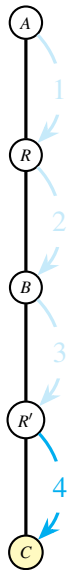
○ Node

Classical packet forwarding



- Physical path
- Transmission
- Node

Classical packet forwarding



- Physical path
- Transmission
- Node

NetKAT: Semantic Foundations for Networks

Carolyn Jane Andersson
Swansea 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 standardized 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 im-

* 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. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@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://dx.doi.org/10.1145/2593836.2593862

NetKAT: Semantic Foundations for Networks

Carolyn Jane Anderson
Swanmore College *

Nate Foster
Cornell University

Arijun Guha

Jean-Baptiste Jeannin
Carnegie Mellon University *

Dexter Kozen
Cornell University

Cole Schlie
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

*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. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@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://dx.doi.org/10.1145/2538036.2539062

implemented w standard interfa the 1970s: out e firewalls, load with custom h faces. This des functionality a their behavior.

However, i software-defi troller mucho The controlle tions from b re-programm has a global ment a wide routing, traf plicated i and fault-s

A major any vendor clearly spe and define tions. How as OpenFl but diffi

Network, ferent re for SDN gressions work, pr thereby applicat project purpose forward to a nu accessig can be a sequ are wo marts torse St guage plu li

Probabilistic NetKAT

Nate Foster¹, Dexter Kozen¹, Konstantinos Mamouras^{2*}, Mark Reitblatt³, and Alexandra Simu⁴

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 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 like Flowlog [32], P4 [12], Pyretic [36], 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 [23], 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-forwarding functions. This approach, while well enough in settings where 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

Ariun Gubs

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 equational theory. We describe primitives for filtering, modifying,

implemented with a standard interface in the 1970s: out of firewalls, load balancers, and proxies with custom interfaces. This design provides the functionality and flexibility of their behavior.

However, a software-defined controller machine. The controller functions from hardware re-programmable has a global memory a wide routing, traffic sophisticated and fault-tolerant.

A major vendor clearly specifies and defines them. How is OpenFl but difficult

D.3.2 [Programming Language] Specialized application lan-

'ng, Frenetic, Network pro-
languages. Kleene algebra

Network

for SDN
gratuitin
work pr
thereby
applicat
project
purpose
forward
to a fut

alled "the last bastion of
n computers, which are

of this work for personal or
are not made or distributed
notice and the full citation
owned by others than ACM
79 otherwise, or republica-
with permission, and the

message
can be
a sequ
are w
man's
format
St
guag
ele li

Temporal NetKAT

Ryan Beckett
Princeton University, USA
beckett@princeton.edu

Michael Greenberg
Pomona College, USA
mgreenb@pomona.edu

David Walker
Princeton University, USA
walker@princeton.edu

1. Introduction

[illegible]

Probabilistic NetKAT

Nate Foster¹, Dexter Kosen¹, Konstantinos Mamouras^{2*},
Mark Reithblatt^{3*}, and Alexandra Sibley⁴

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 the semantics from one based on deterministic behaviors and 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 *Proteus* [11], *Prover* [36], *Maple* [32], *Flowlog* [38], and others are enabling programmers to specify the intended behavior of a network in terms of enabling programmers to specify the predicates and functions on packets. Verification tools like *Booster* *Spec 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 few shortcomings. They model network behavior in terms of deterministic packet forwarding functions. This approach works well in cases where the packet forwarding function is simple, or where the proportion of interest only accounts of 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.

^a Work performed at Cornell University.

NetKAT: Semantic Foundations for Networks

Ariun Guba

Cole Schles
Princeton Un.

Abstract

implemented w
standard interfa
the 1970s: out o
firewalls, load
with custom h
faces. This des
functionality a
their behavior.

However, i
software-defin
troller machi
The controlle
tions from h
re-programm
has a global
ment a wide
routing, tra
phisticated i
and fault-to

A major
any vendor
clearly spe
and defin
tions. How
as OpenFI
but diffic

Michael Greenberg
Pomona College, USA
mgreenb@pomona.edu

David Walkott
Princeton University, USA
walkott@princeton.edu

[illegible][illegible]

ing, Frentic, Network pro-
languages, Kleene algebra

Network
ferent re
for SDN
grammar
work pro
thereby
applicat
project
purpose
forward
to a run
messag
can be
a sequ
are wo
manip
to trust
St
guage
ele li

of this work for personal or
are not made or distributed
notice and the full citation
used by others than ACM
79 otherwise, or republish,
with permission and the

Nate Foster¹, Dexter Koenig³, Konstantinos Mamouras^{2*},
Mark Reithblatt^{3*}, and Alexandra Siles⁴

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.

Formal specification and verification of networks have become a reality in recent years with the convergence of network-specific hardware to a general purpose-chip, allowing the use of programming languages like Prolog [12] or C [13]. Maple [12], Flex [13], and other programming languages like Prolog [12] or C [13], are making programmers to model the behavior of a network in terms of high-level constructs such as *Boolean* predicates and *Boolean* expressions on packets. Verifications such as *Boolean* satisfiability [21], VeriPro [22], and other SAT [23] are making it possible to check the properties such as connectivity, loop freedom, and traffic isolation automatically. However, even though many network analysis frameworks all have a finite reasoning function. This approach works well enough for a determination of network functionality, but it is not clear, or where the properties of interest are the forwarding paths used to carry traffic. If it does not provide sufficient accounts of more complicated situations that arise in practice:

Congestion: the network operator wishes to calculate the expected congestion on each link given a model of the demands for the network.

Pathloss: the network operator wishes to calculate the probability of a packet being delivered to its destination, given that devices are subject to a certain probability.

* Work performed at Cornell University.

This work presents *theANe*, a functional and
NeKAT (PNE). Like PNE, it enables
choice and infinite iteration
and higher-order
The 6

[illegible]

NetKAT: Semantic Foundations for Networks

Ariun Guba

Cole Schles
Princeton Un.

Probabilistic NetKAT

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 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 a studies that show how the language can be used to model a diverse collection of scenarios drawn from real-world networks.

Temporal NetKAT

David Walklett
Princeton University, US
dwalklett@princeton.edu

DyNetKAT: An Algebra of Dynamic Networks *

tunc@cs.au.dk

1. Introduction

1. Introduction

In software-defined networking, a general-purpose router, or cluster of machines, must deliver massive, programmable flows of data and control APIs with low overhead. This is a challenging task for current network operating systems (OSs) because they require a complex, multi-tiered architecture to support a wide range of key technologies, including the most recent forwarding and scheduling algorithms, traffic patterns and sampling rules. Over the past decade, SDN vendors have been tackling it in a variety of ways. For example, Flowlogix [1] uses a multi-tier architecture, while P4 [2] and OpenFlow [3] use a single-tier architecture. In this paper, we present a new language for programming SDN routers that leverages the strengths of both [2] and [3]. OpenFlow [3] provides a simple, declarative programming model for configuring and managing network devices, while P4 [2] provides a more expressive, imperative programming model for configuring and managing network devices. In this paper, we present a new language for programming SDN routers that leverages the strengths of both [2] and [3].

Abstract. We introduce a formal language for specifying dynamic updates for Software Defined Networks. Our language builds upon Network Kiosque Algebra with Tests (NetKAT) and adds constructs for synchronisations and multi-point 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 DyNetKat – 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.

duction

diffusion and verification of networks has become a reality in networking tools. Programming-language-specific programming languages and frameworks such as *FlowCirc* [3] and others are enabling programmers to specify the network flow in terms of high-level declarative programming languages to specify the network flow. Verification tools like *FlowChecker* [4] and *Booster* [5], and *NetATM* [6] are making it possible to check specific activity, loop freedom, and traffic isolation automatically. Despite many notable advances, network verification automatically analyzing network behavior in terms of frameworks all have a fundamental limitation. The approach works well enough to estimate probabilities in simple cases where the properties of interest only need a few paths used to carry traffic. The properties of interest only need no complicated situations that often arise in practice.

Our approach is to let the network operator write in a declarative manner on the network operator wishes to calculate the expected behavior on each link given a model of the demands for traffic. The network operator wishes to calculate the probability of traffic delivered to their destination, given that device, and the probability of any probability.

PLoNK: Functional Probabilistic NetKAT
ALEXANDER VANDENBROUCKE, KU Leuven, Belgium
TOM SCHRIJVERS, KU Leuven, Belgium
This work presents PLoNK, a framework for reasoning about probabilistic network properties and analyzing their complexity.

[illegible]

NetKAT: Semantic Foundations for Networks

Carolyn Jane Anderson
Swanthon College *

Nate Foster
Cornell University

Ariun Guha

Concurrent NetKAT Modeling and analyzing statel, concurrent networks

Jana Wagners¹ , Nate Foster² , Tobias Kopp³ ,
Dexter Kozen⁴ , Jurriaan Rut¹, and Alexandra Silva² 

¹ Radboud University, Nijmegen, The Netherlands
Jana.Wagners@cs.ru.nl
² Cornell University, Ithaca, New York, USA
³ ILIC, University of Amsterdam, The Netherlands

Jean-Baptiste Jeannin
Carnegie Mellon University *

Dexter Kozen
Cornell University

Cole Schlie
Princeton Un

Probabilistic NetKAT

Nate Foster¹, Dexter Kozen², Konstantinos Mamouras^{2*},
Mark Reitblatt^{3*}, and Alexandra Silva⁴

¹ Cornell University
² University of Pennsylvania
³ Bedford
⁴ University College London

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, complete equational theory. We describe primitives for filtering, modifying, and sequential composition of that iterates programs. We show how to model and reason about networks with tests (KAT) and prove

implemented w standard interfa the 1970s: out c firewalls, lead with custom h faces. This des functionality a their behavior. However, i software-defic troller much the control tions from b re-programm has a global ment a wide

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 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 a study that shows how the language can be used to model a diverse collection of scenarios drawn from real-world networks.

Introduction

Verification and verification of networks has become a reality in re- sulting tools. Programming languages like *Formal* [11], *Pyrite* [36], *FlowLog* [38], and others are enabling programmers to specify the behavior of a network in terms of high-level constructs such as *Flow* [22], and *NetKAT* [12] are making it possible to check properties despite more network advances, these frameworks all have a fun- tion: they model network behavior in terms of deterministic per- spective. This approach, works well enough in settings where g paths need to carry traffic. But it does not provide exist- on the network operator wishes to calculate the expected on on each link given a model of the demands for traffic delivered to their destination, given that devices and on its probability.

at Cornell University.

PLoINK: Functional Probabilistic NetKAT

ALEXANDER VANDENBROUCKE¹, RIZ LARSEN², Benjamin TOM SCHRIEVERS¹, LIO LARSEN²

This work presents PLoINK, a functional probabilistic network programming language that extends Probabilistic NetKAT (PNK). Like PNK, it enables probabilistic modeling of network behaviors by providing probabilistic choice and higher-order functions to make programming much more convenient. The formalization of PLoINK challenges the long-standing view that probabilistic languages are too complex to be used in practice. We believe that one must understand the theory of the language and its semantics before it can be used in practice.

Keywords: Concurrent Kleene algebras, NetKAT, completeness, concurrency

1 Introduction

Kleene algebras (KA) is a well-studied formalism [29,23,34,8] 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) [21] model conditional control flow while NetKAT [12,10] models behaviors 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 packets. Meanwhile, the software-defined network (SDN) paradigm including data aggregation, richer functionality based on stateful processing including data aggregation and dynamic routing. In languages like the P4 [4], tests of concurrency arise because the semantics depends on the order that packets are processed.

Given this context, it is natural to wonder how we can add concurrency to NetKAT while retaining the elegance of the underlying framework. However, to do so this question is the affirmative, by developing ChieNetKAT. However, to do so this question is the affirmative, by developing ChieNetKAT. However, to do so this question is the affirmative, by developing ChieNetKAT.

© The Author(s) 2022
1. Springer (Ed.) 2022. ISBN 3300 13240 0. pp. 375-400, 2022.
https://doi.org/10.1007/978-3-030-89008-3_31

These two cases together allow programmers to write down concise properties of a packet's path through the network and to make dynamic packet forwarding, as well as control or adapt the programming. The combined equations to being useful for programming. The combined equations to being useful for programming. The combined equations to being useful for programming.

DiNetKAT: An Algebra of Dynamic Networks *

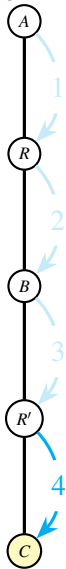
Georgina Calta¹ , Hossein Hojati² , Mohammad Reza Mousavi¹ , and Hilarin Cui Ting³

¹ University of Konstanz, Germany & University of Twente, The Netherlands
g.e.c.calta@uni-konstanz.de
² TIAS, Kluhan University & University of Tehran, Iran
hojati@khu.ac.ir
³ King's College London, UK
mohammad.reza.mousavi@kcl.ac.uk
⁴ University of Konstanz, Germany & Aarhus University, Denmark
cui@kcl.ac.uk

Abstract. We introduce a formal language for specifying dynamic updates for Software Defined Networks. Our language builds upon Network Kleene Algebras with Tests (NetKAT) and adds constructs for synchronization 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 DiNetKAT – 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.

Konstantinos S. Bouts, Defining Networks, Dynamic Updates, Dynamic

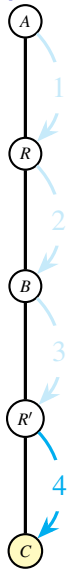
NetKAT expression p^* is encoding network's behavior



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

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

NetKAT expression p^* is encoding network's behavior

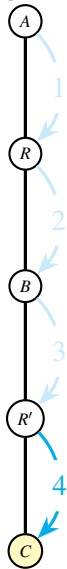


test

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

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

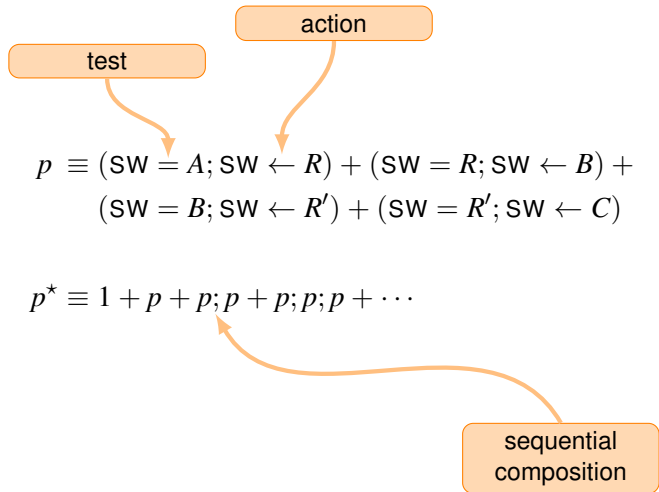
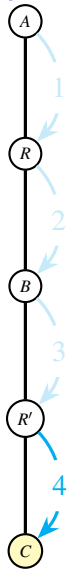
NetKAT expression p^* is encoding network's behavior



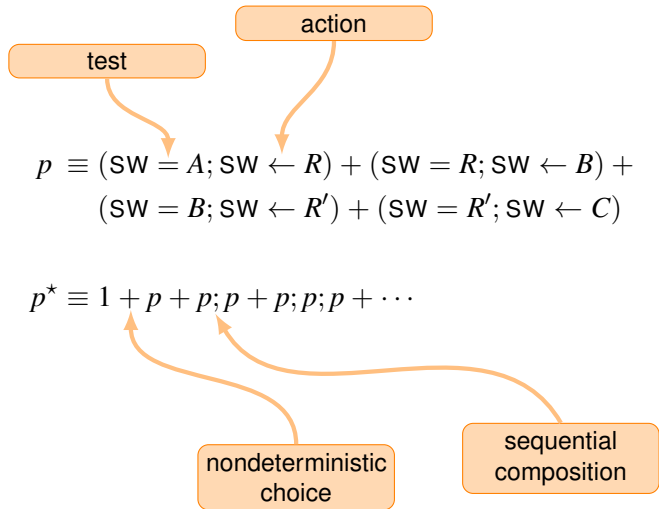
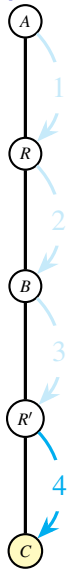
$$p \equiv (\text{SW} = A; \text{SW} \leftarrow R) + (\text{SW} = R; \text{SW} \leftarrow B) + \\ (\text{SW} = B; \text{SW} \leftarrow R') + (\text{SW} = R'; \text{SW} \leftarrow C)$$

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

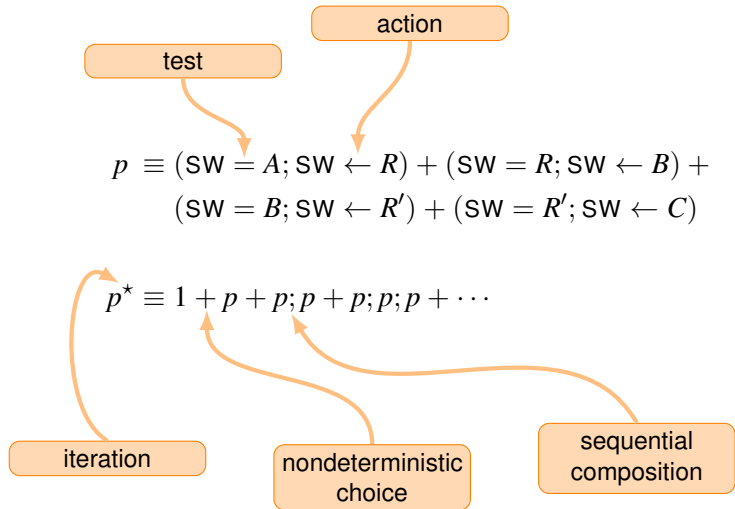
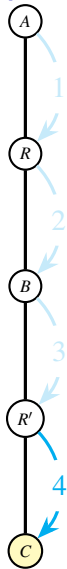
NetKAT expression p^* is encoding network's behavior



NetKAT expression p^* is encoding network's behavior



NetKAT expression p^* is encoding network's behavior



Artwork by Sandbox Studio, Chicago with Ana Kova



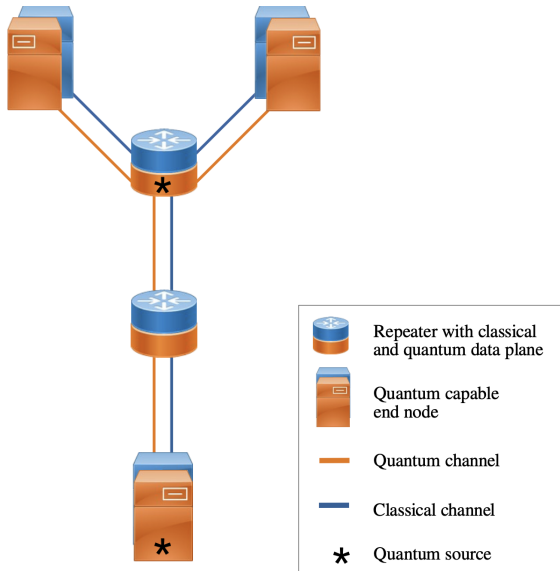
'spooky action at a distance'



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

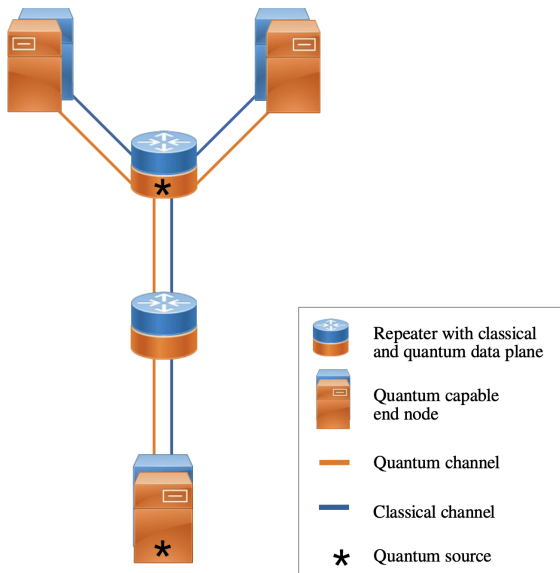




Quantum network:
providing *communication services* to distributed quantum applications

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

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



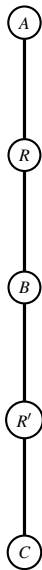
Quantum network:
providing *communication services* to distributed quantum applications

end-to-end Bell pair distribution²

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

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

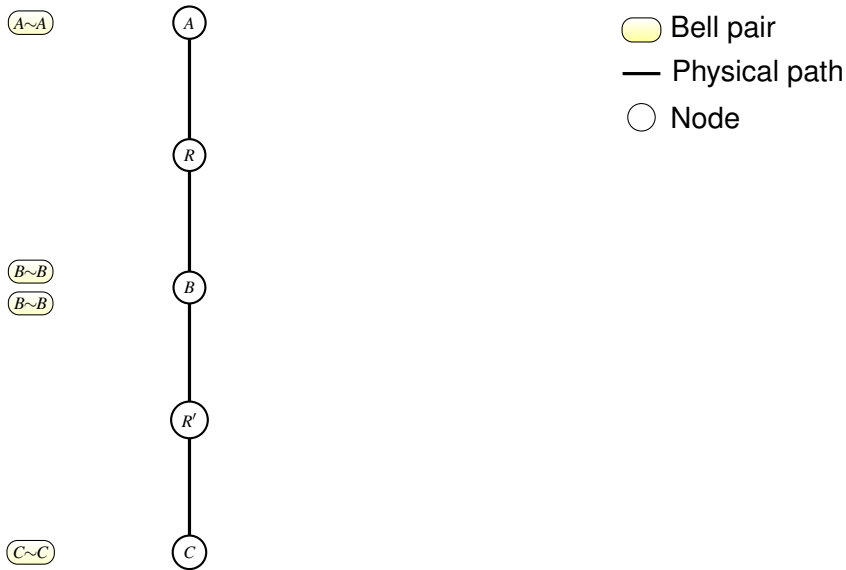
Bell pair generation



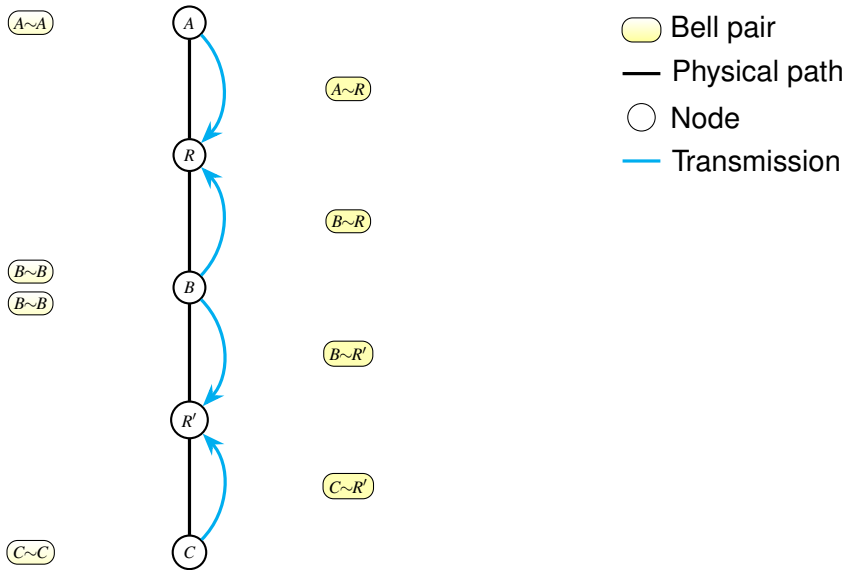
— Physical path

○ Node

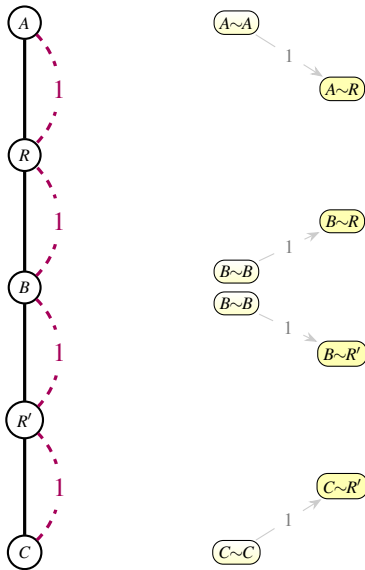
Bell pair generation: Protocol I



Bell pair generation: Protocol I

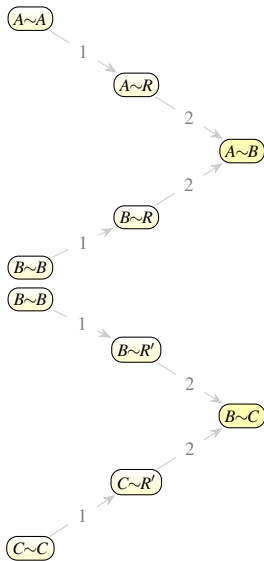
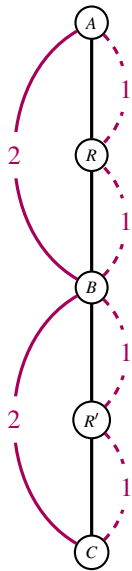


Bell pair generation: Protocol I, round 1



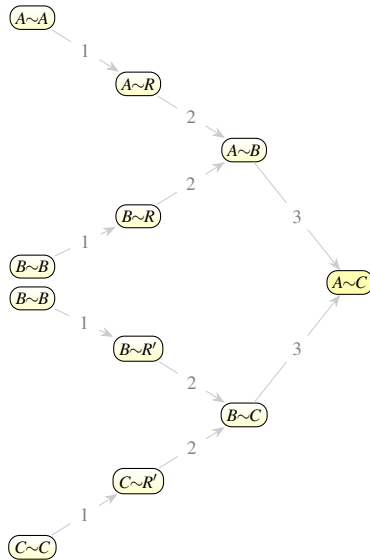
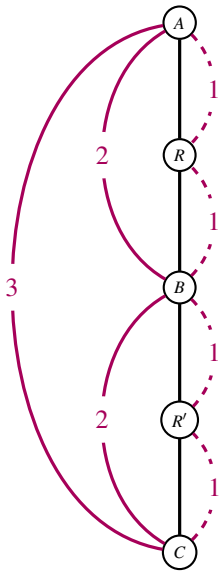
-  Bell pair
-  Physical path
-  Node
-  Transmitted link

Bell pair generation: Protocol I, round 2



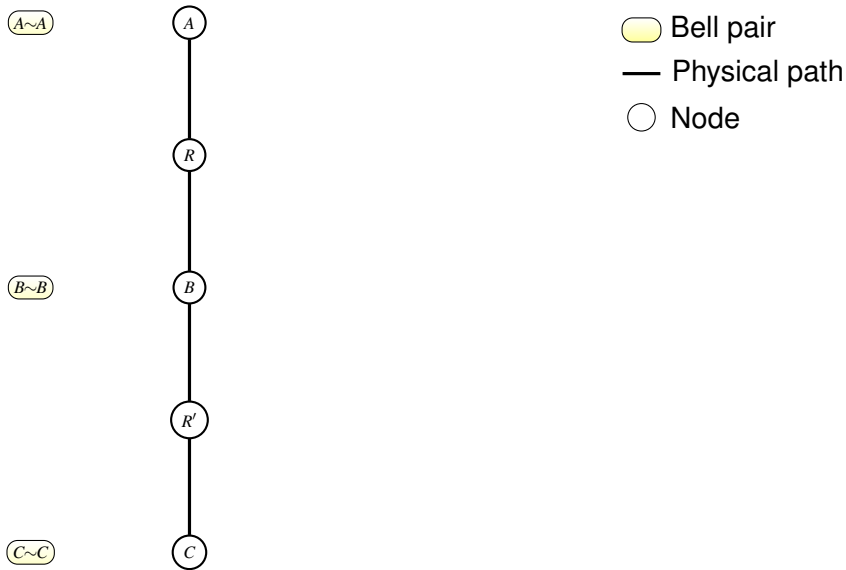
-  Bell pair
-  Physical path
-  Node
-  Transmitted link
-  Swapped link

Bell pair generation: Protocol I, round 3

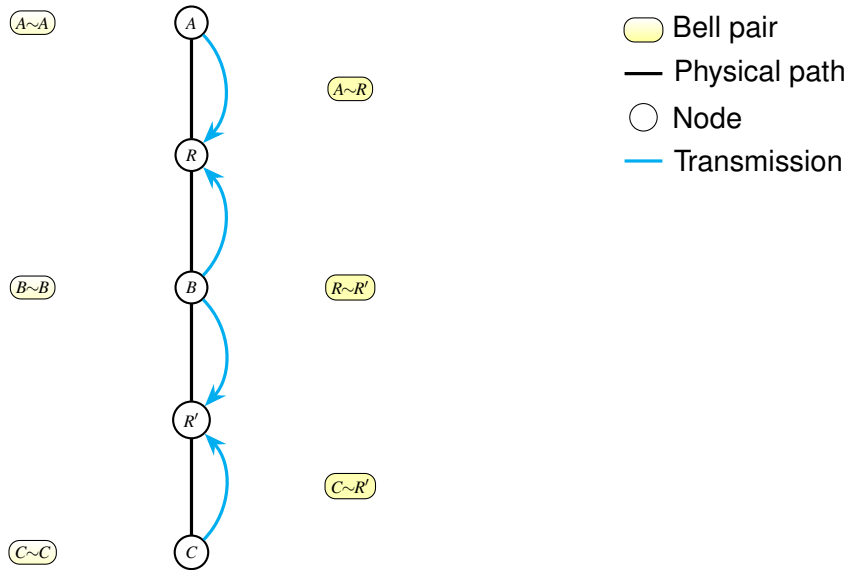


- Bell pair
- Physical path
- Node
- Transmitted link
- Swapped link

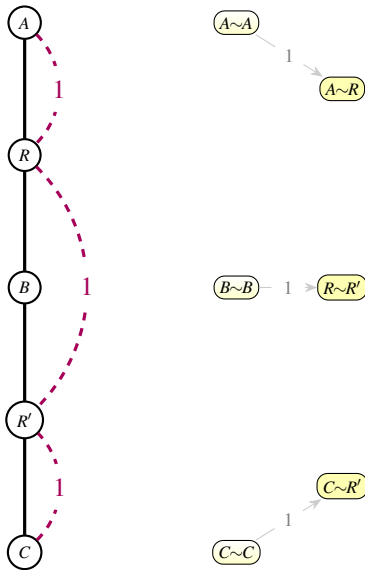
Bell pair generation: Protocol II



Bell pair generation: Protocol II

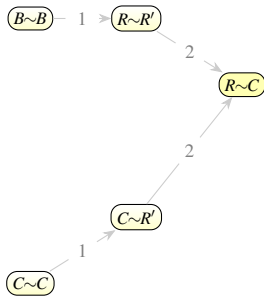
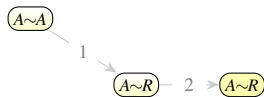
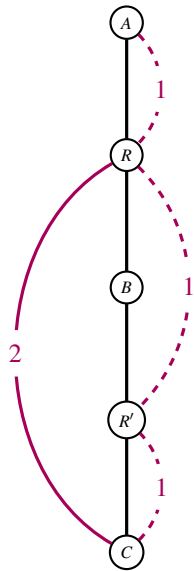


Bell pair generation: Protocol II, round 1



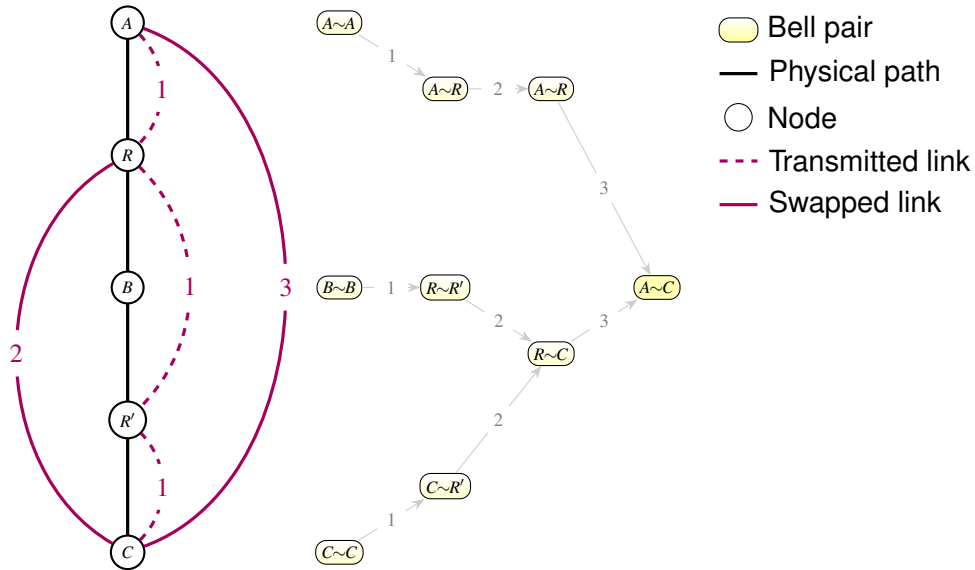
-  Bell pair
-  Physical path
-  Node
-  Transmitted link

Bell pair generation: Protocol II, round 2

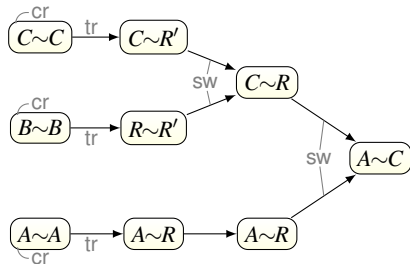
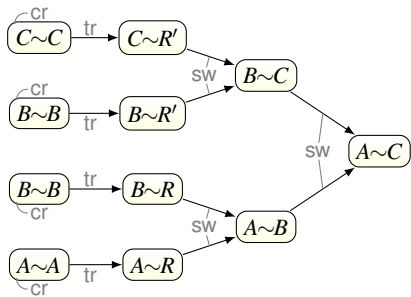


-  Bell pair
-  Physical path
-  Node
-  Transmitted link
-  Swapped link

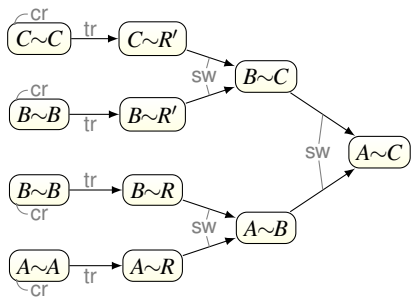
Bell pair generation: Protocol II, round 3



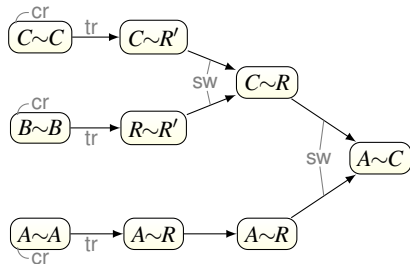
Specification: Protocol I and Protocol II generating $A \sim C$



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



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



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

Basic action $r \triangleright o$

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

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

$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$

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 $r \triangleright o$

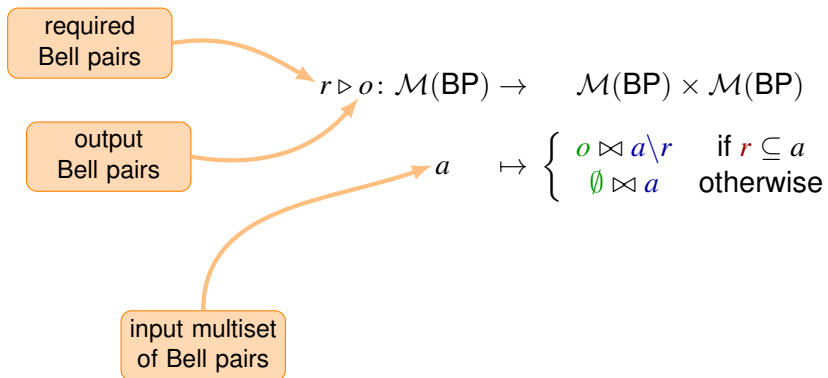
required
Bell pairs

output
Bell pairs

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

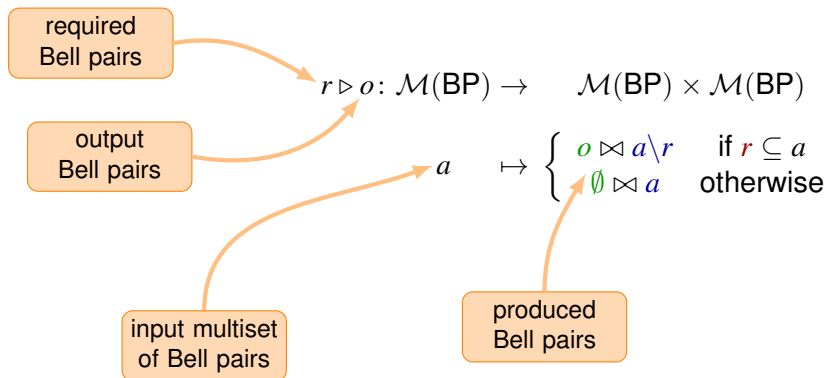
$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$



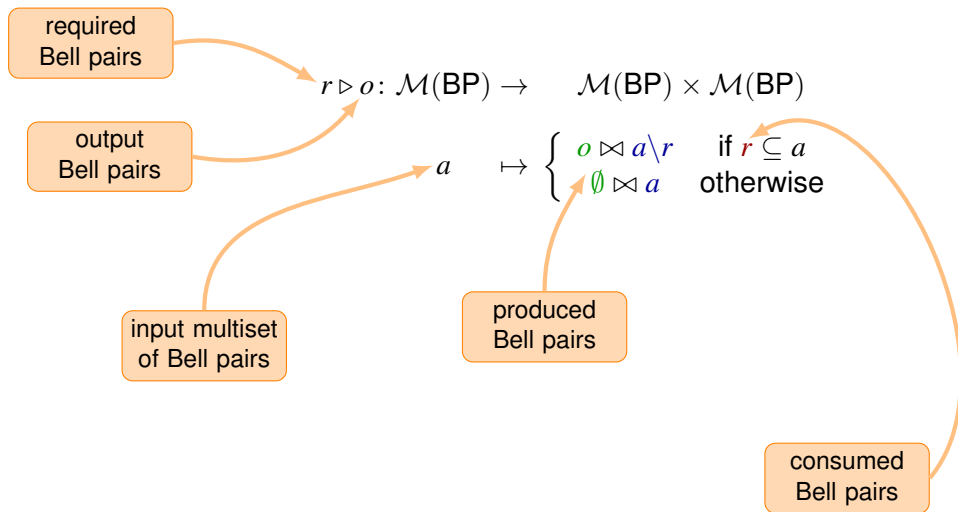
$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$



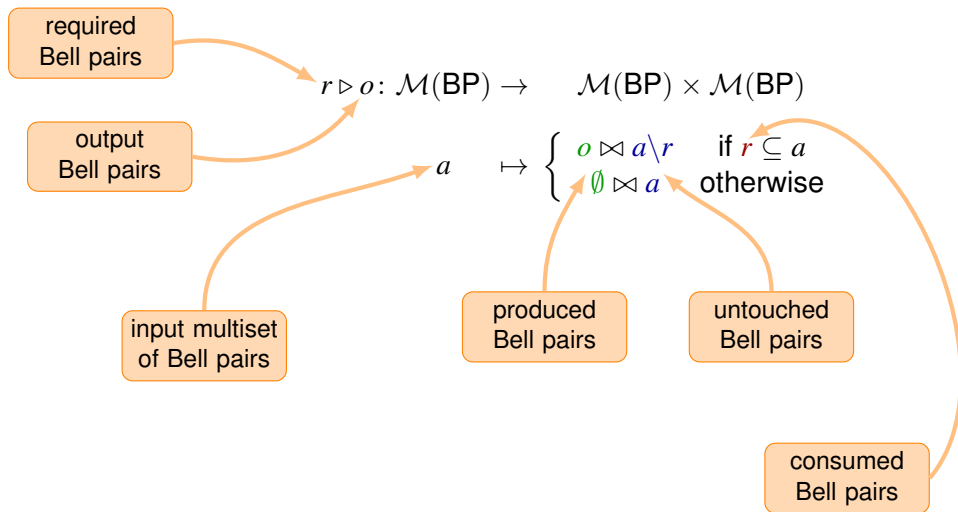
$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$



$\bowtie$: pair of multisets of Bell pairs

Basic action $r \triangleright o$



$\bowtie$: pair of multisets of Bell pairs

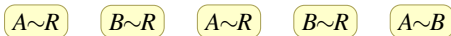
Basic action

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

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

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

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



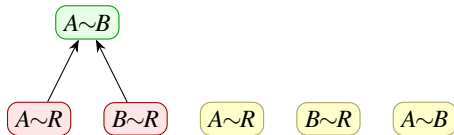
Input Bell pairs

Basic action

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

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

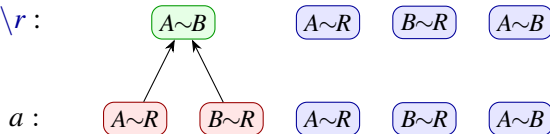
$$r \triangleright o: \mathcal{M}(\text{BP}) \rightarrow \mathcal{M}(\text{BP}) \times \mathcal{M}(\text{BP})$$

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

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

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

$o \bowtie a \setminus r:$



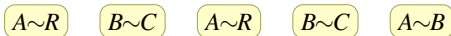
Bell pairs: **consumed**, **produced** and **untouched**

Basic action

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

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

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

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

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

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

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

$\emptyset \bowtie a:$

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

$a:$

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

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

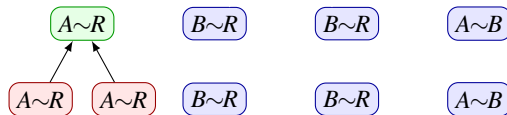
$$r \triangleright o + r \triangleright \emptyset \triangleq r \triangleright o + \text{fail}\langle r \rangle$$

Probabilistic basic actions

Example (Distillation is inherently probabilistic.)

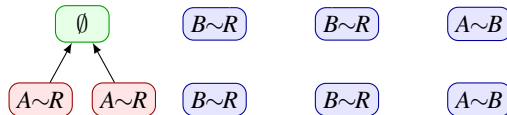
$$\text{di}\langle A \sim R \rangle \triangleq \{A \sim R, A \sim R\} \triangleright \{A \sim R\} + \{A \sim R, A \sim R\} \triangleright \emptyset$$

succeed :



input a :

fail :



input a :

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

Protocols

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

Protocols

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

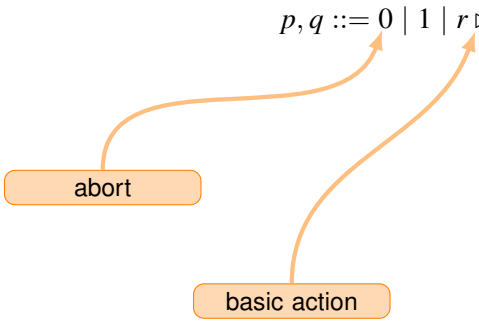


basic action

Protocols

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

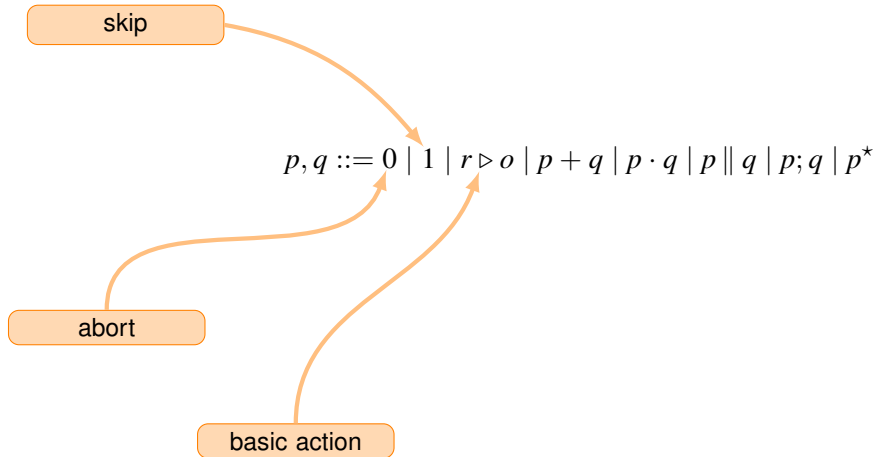
abort



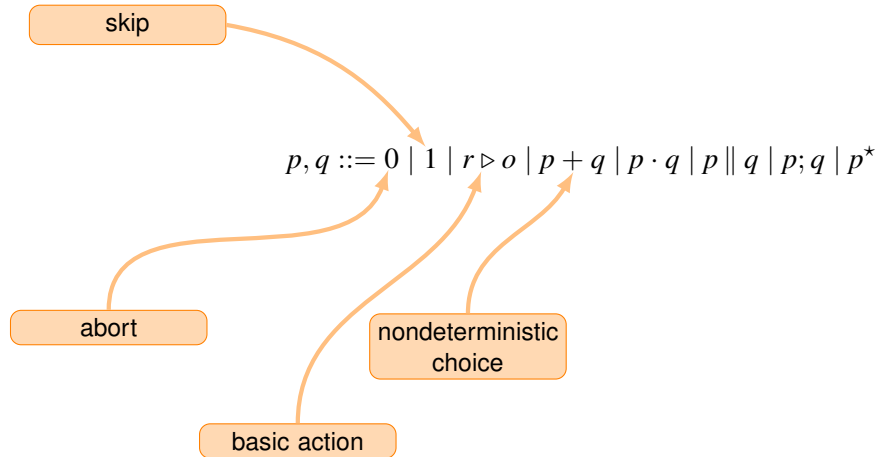
The diagram consists of two orange rounded rectangular boxes. The first box, labeled 'abort', has an arrow that curves upwards and to the right, pointing to the 'r \triangleright o' part of the protocol definition. The second box, labeled 'basic action', has an arrow that curves upwards and to the right, also pointing to the 'r \triangleright o' part of the protocol definition.

basic action

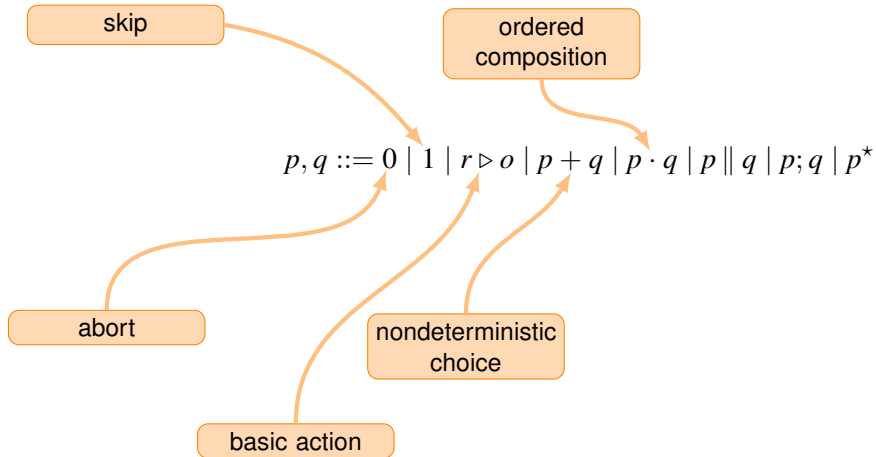
Protocols



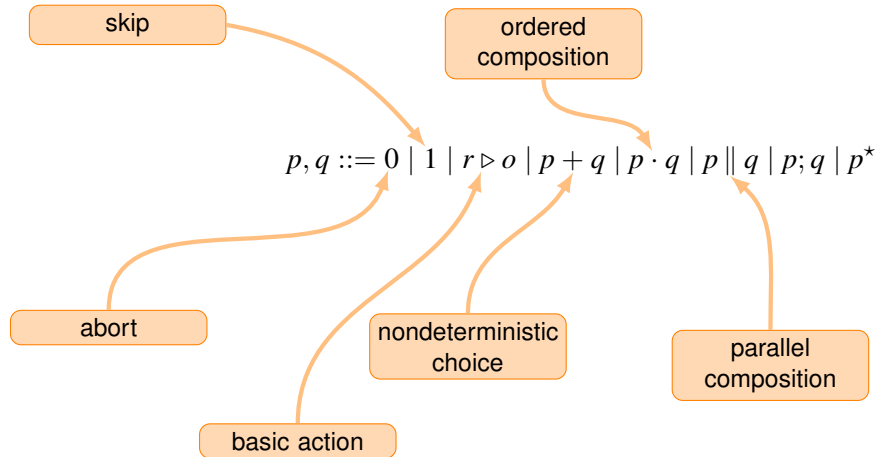
Protocols



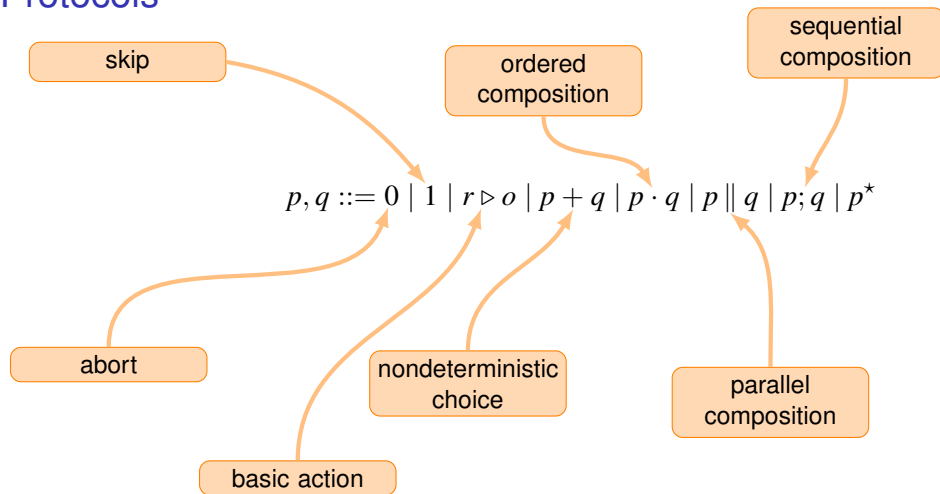
Protocols



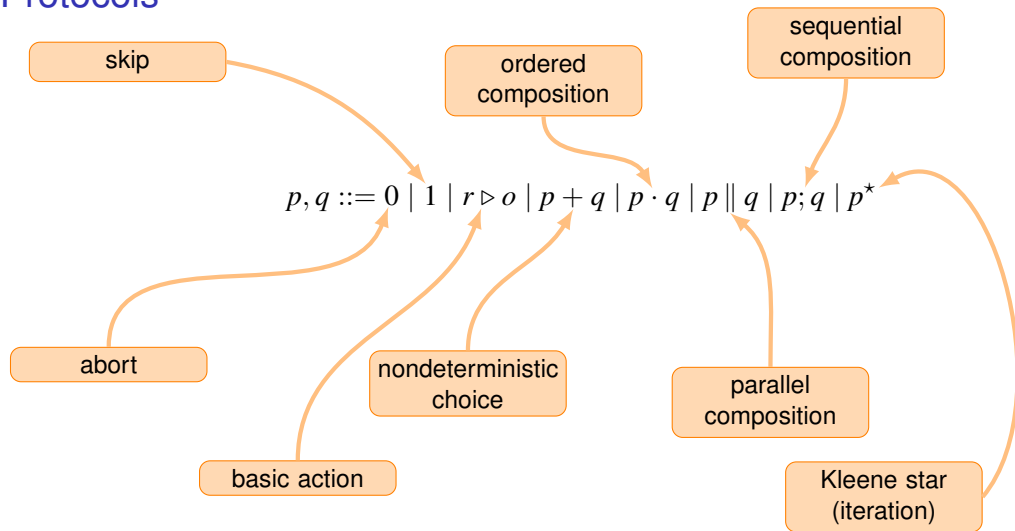
Protocols



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

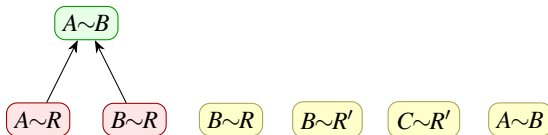


Input Bell pairs

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

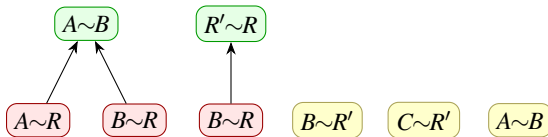


Bell pairs: **input** and **consumed**, **produced**

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

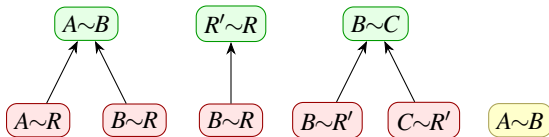


Bell pairs: **input** and **consumed**, **produced**

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

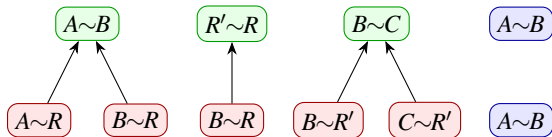


Bell pairs: **input** and **consumed**, **produced**

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

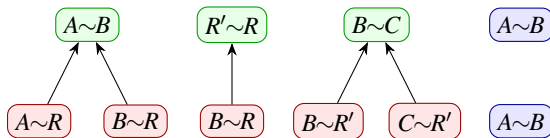


Bell pairs: **consumed**, **produced** and **untouched**

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

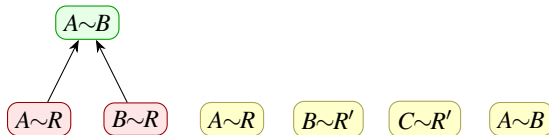
$$\text{sw}\langle A \sim B @ R \rangle \parallel \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \parallel \text{sw}\langle B \sim C @ R' \rangle$$



Bell pairs: input and consumed, produced

Parallel composition vs. ordered composition

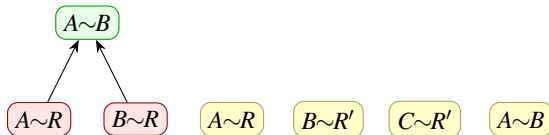
$$\underline{\text{sw}\langle A \sim B @ R \rangle} \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$$



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

Parallel composition vs. ordered composition

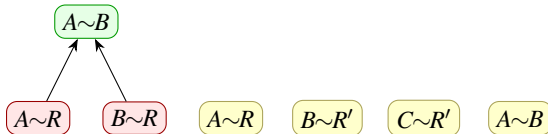
$$\underline{\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle}$$



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

Parallel composition vs. ordered composition

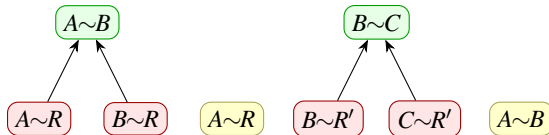
$$\underline{\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle}$$



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

Parallel composition vs. ordered composition

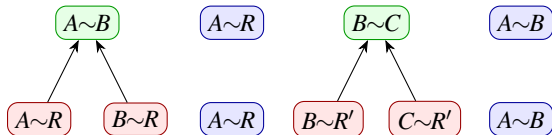
$$\underline{\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle}$$



²Bell pairs: **input** and **consumed**, **produced**.

Parallel composition vs. ordered composition

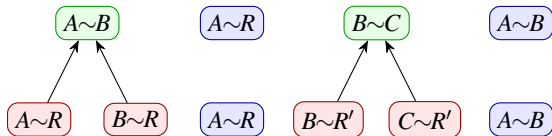
$$\underline{\text{sw}\langle A \sim B @ R \rangle \cdot \text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle}$$



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

Parallel composition vs. ordered composition

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



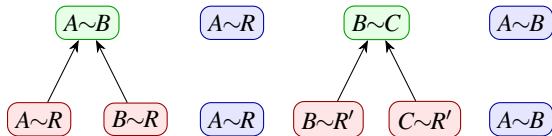
$$\text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle A \sim B @ R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle$$



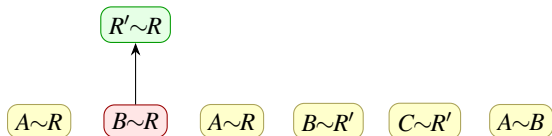
Bell pairs: **consumed**, **produced** and **untouched**

Parallel composition vs. ordered composition

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



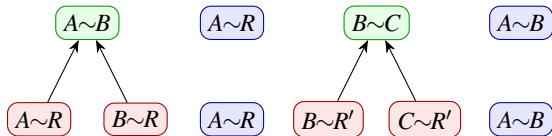
$$\underline{\text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle A \sim B @ R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle}$$



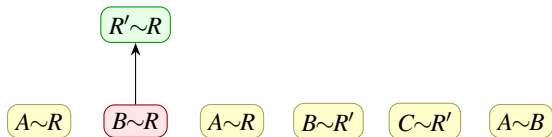
Bell pairs: **consumed**, **produced** and **untouched**

Parallel composition vs. ordered composition

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



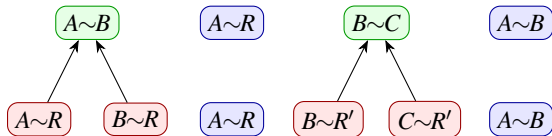
$$\underline{\text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle A \sim B @ R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle}$$



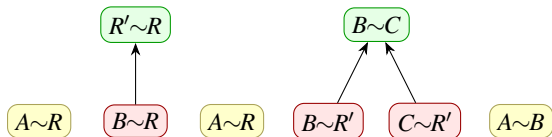
Bell pairs: **consumed**, **produced** and **untouched**

Parallel composition vs. ordered composition

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



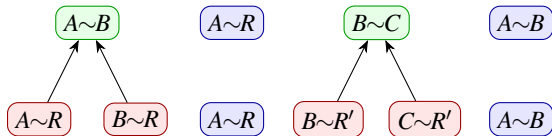
$$\underline{\text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle A \sim B @ R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle}$$



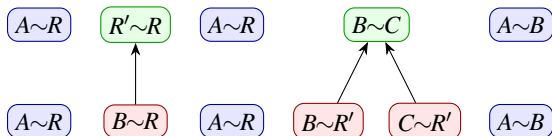
Bell pairs: **consumed**, **produced** and **untouched**

Parallel composition vs. ordered composition

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



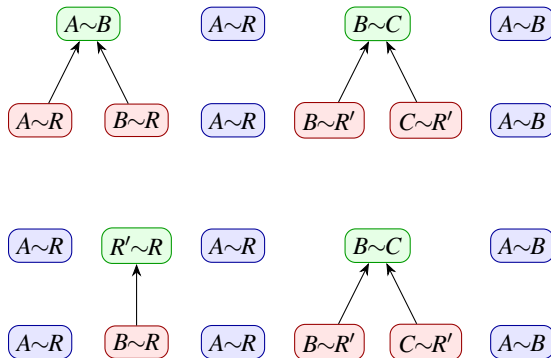
$$\underline{\text{tr}\langle B \sim R \rightarrow R' \sim R \rangle \cdot \text{sw}\langle A \sim B @ R \rangle \cdot \text{sw}\langle B \sim C @ R' \rangle}$$



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

Parallel composition vs. ordered composition

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



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

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!

