A Unified Characterization of Completeness and Triviality for Secure Function Evaluation

Hemanta K. Maji
Manoj Prabhakaran
Mike Rosulek

A Unified Characterization of Completeness and Triviality for Secure Function Evaluation

Hemanta K. Maji
Manoj Prabhakaran
Mike Rosulek

A Unified Characterization of Completeness and Triviality for Secure Function Evaluation

Hemanta K. Maji
Manoj Prabhakaran
Mike Rosulek

A Unified Characterization of Completeness and Triviality for semi-honest Secure Function Evaluation

Hemanta K. Maji
Manoj Prabhakaran
Mike Rosulek

Overview

Secure Function Evaluation

Secure Function Evaluation

- Parties have inputs and want to securely evaluate a function of their respective inputs

Secure Function Evaluation

- Parties have inputs and want to securely evaluate a function of their respective inputs
  - Non-triviality: Associated notion of security

Secure Function Evaluation

- Parties have inputs and want to securely evaluate a function of their respective inputs
  - Non-triviality: Associated notion of security
- Two party case:

Secure Function Evaluation

- Parties have inputs and want to securely evaluate a function of their respective inputs
  - Non-triviality: Associated notion of security
- Two party case:
  - Alice has input `x` and Bob has input `y`

Secure Function Evaluation

- Parties have inputs and want to securely evaluate a function of their respective inputs
  - Non-triviality: Associated notion of security
- Two party case:
  - Alice has input `x` and Bob has input `y`
  - They want to evaluate a function `f`

Secure Function Evaluation

- Parties have inputs and want to securely evaluate a function of their respective inputs
  - Non-triviality: Associated notion of security
- Two party case:
  - Alice has input x and Bob has input y
  - They want to evaluate a function f
    - In general, f provides $f_A(x,y;R)$ to Alice and $f_B(x,y;R)$ to Bob

Secure Function Evaluation

- Parties have inputs and want to securely evaluate a function of their respective inputs
  - Non-triviality: Associated notion of security
- Two party case:
  - Alice has input x and Bob has input y
  - They want to evaluate a function f
    - In general, f provides $f_A(x,y;R)$ to Alice and $f_B(x,y;R)$ to Bob
    - Special cases: $f_A = f_B$; $f_A = \text{const.}$; $|R| = 1$

Security Models

Security Models

- Semi-honest Adversaries

Security Models

- Semi-honest Adversaries
  - Parties follow the protocol but are "curious" to learn additional information

Security Models

- Semi-honest Adversaries
  - Parties follow the protocol but are "curious" to learn additional information
- Active Adversaries

Security Models

- Semi-honest Adversaries
  - Parties follow the protocol but are "curious" to learn additional information
- Active Adversaries
  - Parties behave arbitrarily

Security Models

- Semi-honest Adversaries
  - Parties follow the protocol but are "curious" to learn additional information
- Active Adversaries
  - Parties behave arbitrarily
  - For example: Standalone and UC

Hybrids

Hybrids

- What is g-hybrid?

Hybrids

- What is g-hybrid?
  - Trusted Copy of g is provided in addition to communication channels

Hybrids

- What is g-hybrid?
  - Trusted Copy of g is provided in addition to communication channels

- Secure protocol for `f` in g-hybrid:

Hybrids

- What is g-hybrid?
  - Trusted Copy of g is provided in addition to communication channels

- Secure protocol for `f` in g-hybrid:
  - Securely implement `f` assuming parties have access to a trusted copy of g

Hybrids

- What is g-hybrid?
  - Trusted Copy of g is provided in addition to communication channels

- Secure protocol for `f` in g-hybrid:
  - Securely implement `f` assuming parties have access to a trusted copy of g
- Reduction: If `f` can be securely implemented in g-hybrid then "`f` reduces to g"

Hybrids

- What is g-hybrid?
  - Trusted Copy of g is provided in addition to communication channels

- Secure protocol for `f` in g-hybrid:
  - Securely implement `f` assuming parties have access to a trusted copy of g
- Reduction: If `f` can be securely implemented in g-hybrid then "`f` reduces to g"
  - Note: Needs an associated notion of security

Complete Functions

Complete Functions

- Intuitively: Most "complicated" functions

Complete Functions

- Intuitively: Most "complicated" functions
  - Similar to notion of completeness in Complexity Theory

Complete Functions

- Intuitively: Most "complicated" functions
  - Similar to notion of completeness in Complexity Theory
- An SFE `g` is complete if every `f` reduces to it

Complete Functions

- Intuitively: Most "complicated" functions
  - Similar to notion of completeness in Complexity Theory
- An SFE `g` is complete if every `f` reduces to it
  - That is: Any `f` has secure protocol in `g`-hybrid

Complete Functions

- Intuitively: Most "complicated" functions
  - Similar to notion of completeness in Complexity Theory
- An SFE `g` is complete if every `f` reduces to it
  - That is: Any `f` has secure protocol in `g`-hybrid
  - Example: Oblivious Transfer

Known Completeness Results

Known Completeness Results

- Only Semi-honest case listed here

Known Completeness Results

- Only Semi-honest case listed here
- Symmetric Deterministic Functions with OR-minor are complete [KILIAN89]

Known Completeness Results

- Only Semi-honest case listed here
- Symmetric Deterministic Functions with OR-minor are complete [KILIAN89]
- If and only if version: [KILIAN91]

Known Completeness Results

- Only Semi-honest case listed here
- Symmetric Deterministic Functions with OR-minor are complete [Kilian89]
  - If and only if version: [Kilian91]
- General Deterministic Functions: [Kraschewski-MullerQuade-11]

Known Completeness Results

- Only Semi-honest case listed here
- Symmetric Deterministic Functions with OR-minor are complete [Kilian89]
  - If and only if version: [Kilian91]
- General Deterministic Functions: [Kraschewski-MullerQuade-11]
- Symmetric Randomized Functions: [KilianOO]

Our Aim for this Talk

Our Aim for this Talk

- Provide a characterization of complete (2-party) secure function evaluation

Our Aim for this Talk

- Provide a characterization of complete (2-party) secure function evaluation

- In this talk: We shall restrict to semi-honest security

Unified Result

Unified Result

- `g` is semi-honest complete if and only if `graph(g)` is NOT a "product graph"

Unified Result

- `g` is semi-honest complete if and only if `graph(g)` is NOT a "product graph"
  - What is "`graph(g)`"?
  - What is "product graph"?

Unified Result

- `g` is semi-honest complete if and only if `graph(g)` is NOT a "product graph"
  - What is "`graph(g)`"?
  - What is "product graph"?
  - Rest of the talk: Understanding this result

Technical Content

Representation of SFE

Representation of SFE

- SFE functionality, when presented with inputs x and y, samples a uniform random string r from the set R and outputs $f_A(x,y;r)$ to Alice and $f_B(x,y;r)$ to Bob

Representation of SFE

- SFE functionality, when presented with inputs x and y, samples a uniform random string r from the set R and outputs $f_A(x,y;r)$ to Alice and $f_B(x,y;r)$ to Bob
- A 2-party SFE is represented using a matrix

Representation of SFE

- SFE functionality, when presented with inputs x and y, samples a uniform random string r from the set R and outputs $f_A(x,y;r)$ to Alice and $f_B(x,y;r)$ to Bob
- A 2-party SFE is represented using a matrix
  - The (x,y)-th entry of the matrix contains the joint output distribution over Alice-Bob outputs corresponding to Alice input x and Bob input y

Views of Parties

Views of Parties

- When Alice feeds `x` to the SFE functionality and receives `w` as output, we represent the tuple `(x,w)` as "Alice View"

Views of Parties

- When Alice feeds `x` to the SFE functionality and receives `w` as output, we represent the tuple `(x,w)` as "Alice View"
- Similarly, for Bob input `y` and his output `z`, the tuple `(y,z)` is "Bob View"

Views of Parties

- When Alice feeds `x` to the SFE functionality and receives `w` as output, we represent the tuple `(x,w)` as "Alice View"
  - Similarly, for Bob input `y` and his output `z`, the tuple `(y,z)` is "Bob View"
- Looking ahead: It will be interesting to consider joint views of parties

Example: OR

Example: OR

	0	1
0	0	1
1	1	1

Example: OR

	0	1
0	0	1
1	1	1

(0,0)

Example: OR

	0	1
0	0	1
1	1	1

(0,0)

(0,1)

Example: OR

	0	1
0	0	1
1	1	1

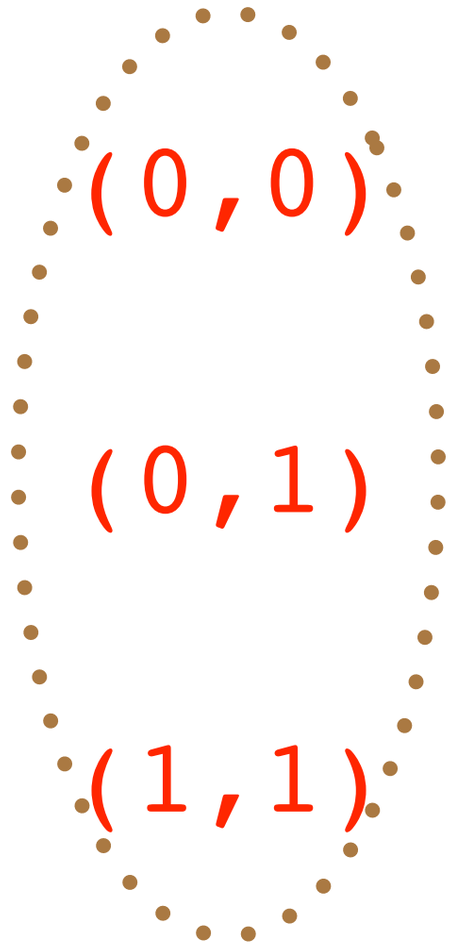
(0,0)

(0,1)

(1,1)

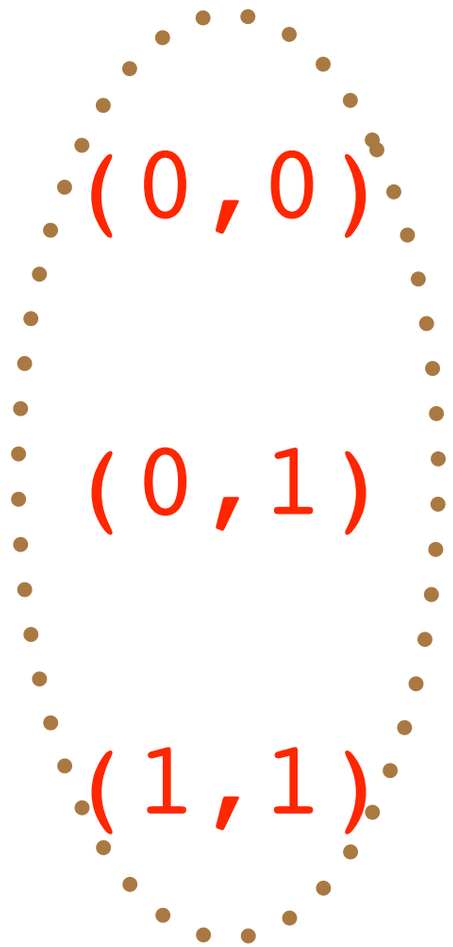
Example: OR

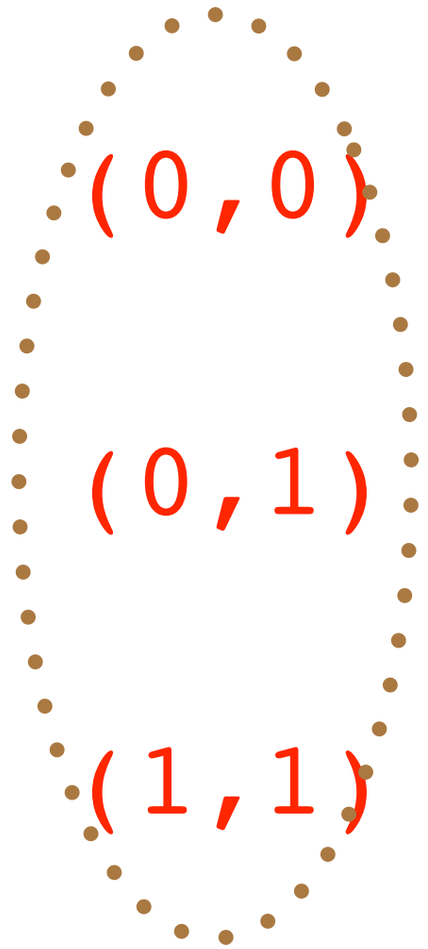
	0	1
0	0	1
1	1	1



Example: OR

	0	1
0	0	1
1	1	1





Example: OR

	0	1
0	0	1
1	1	1

```
(0,0)        (0,0)

(0,1)        (0,1)

(1,1)        (1,1)
```

Example: OR

	0	1
0	0	1
1	1	1

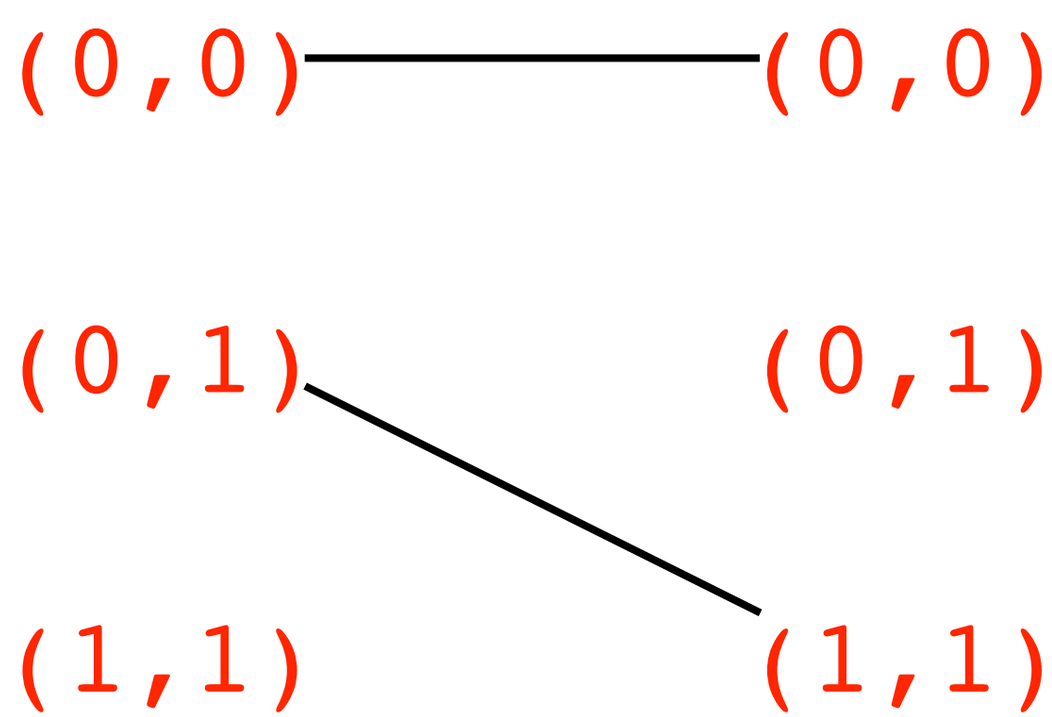
(0,0)——————(0,0)

(0,1) (0,1)

(1,1) (1,1)

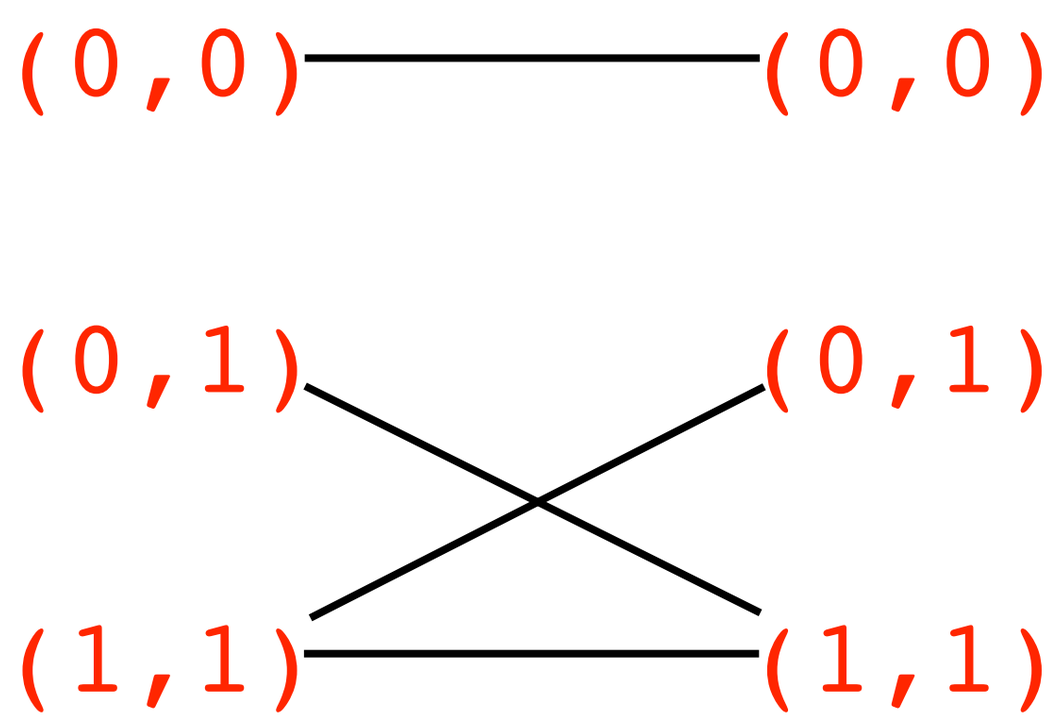
Example: OR

	0	1
0	0	1
1	1	1



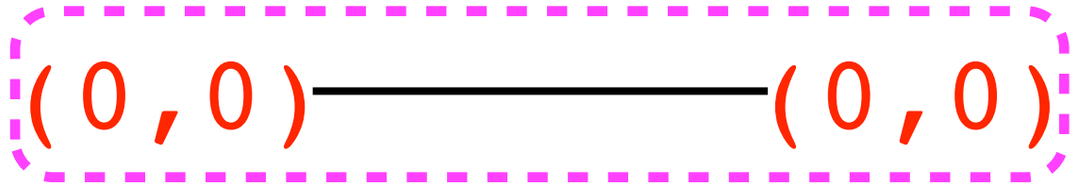
Example: OR

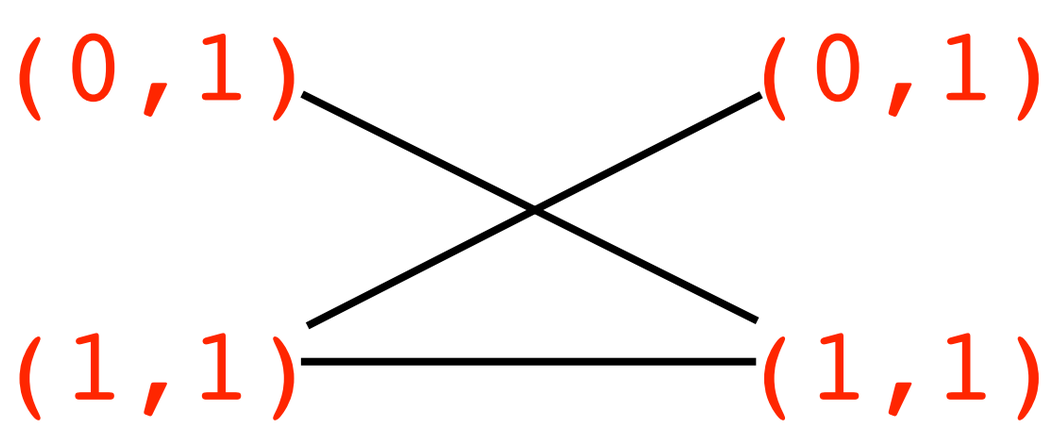
	0	1
0	0	1
1	1	1



Example: OR

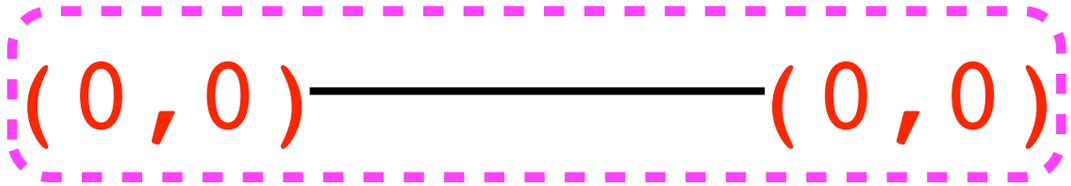
	0	1
0	0	1
1	1	1

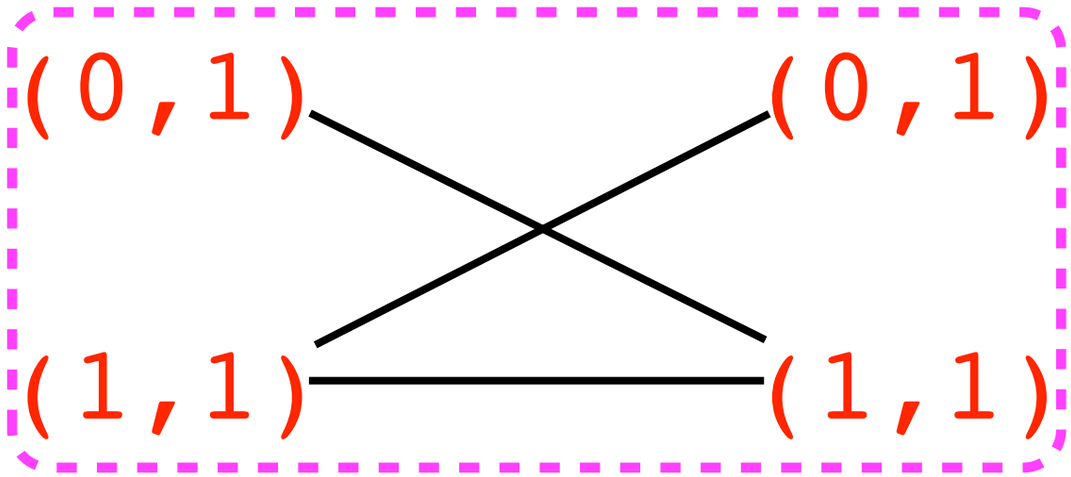




Example: OR

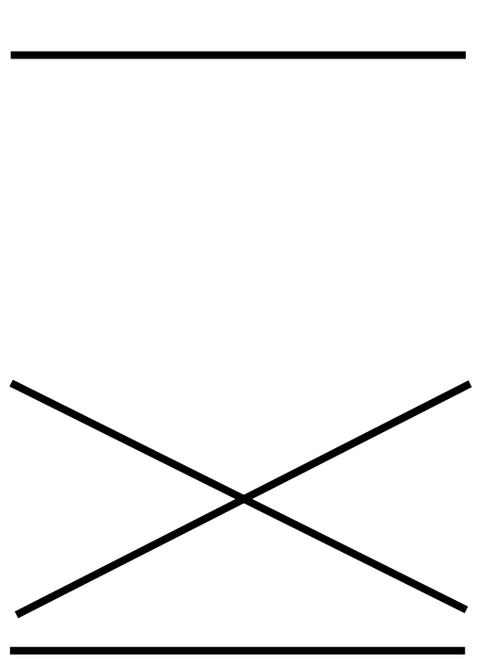
	0	1
0	0	1
1	1	1





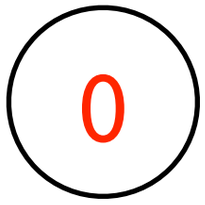
Example: OR

	0	1
0	0	1
1	1	1

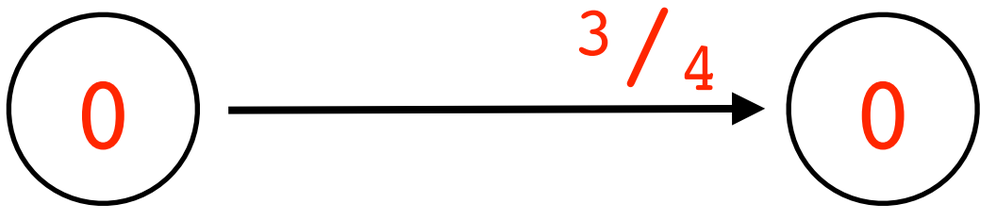


Example: BSC

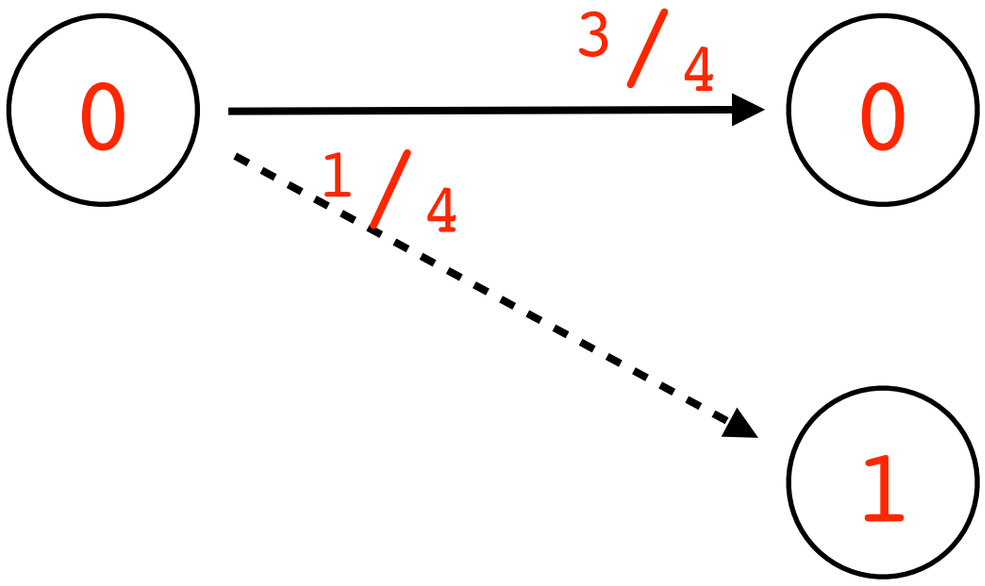
Example: BSC



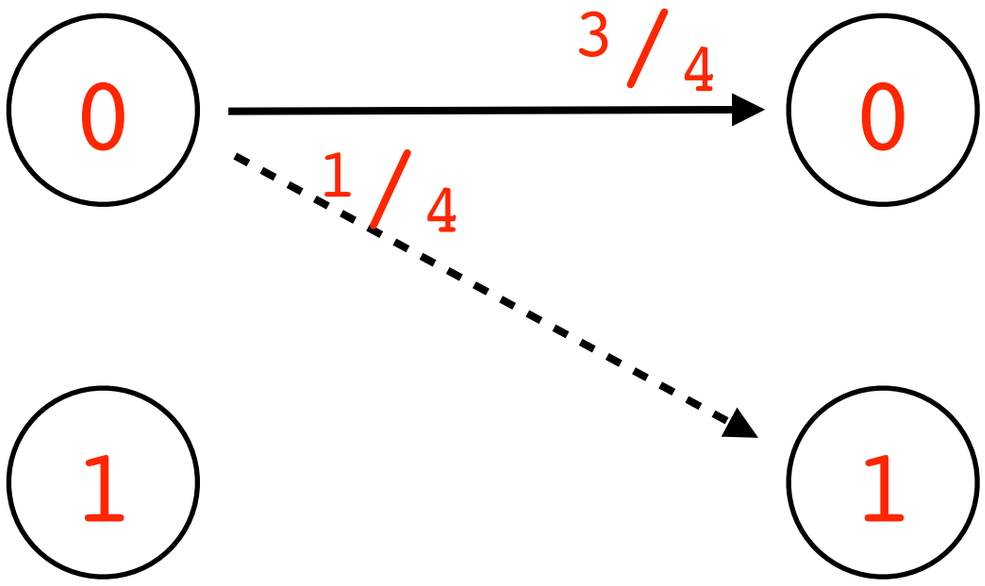
Example: BSC



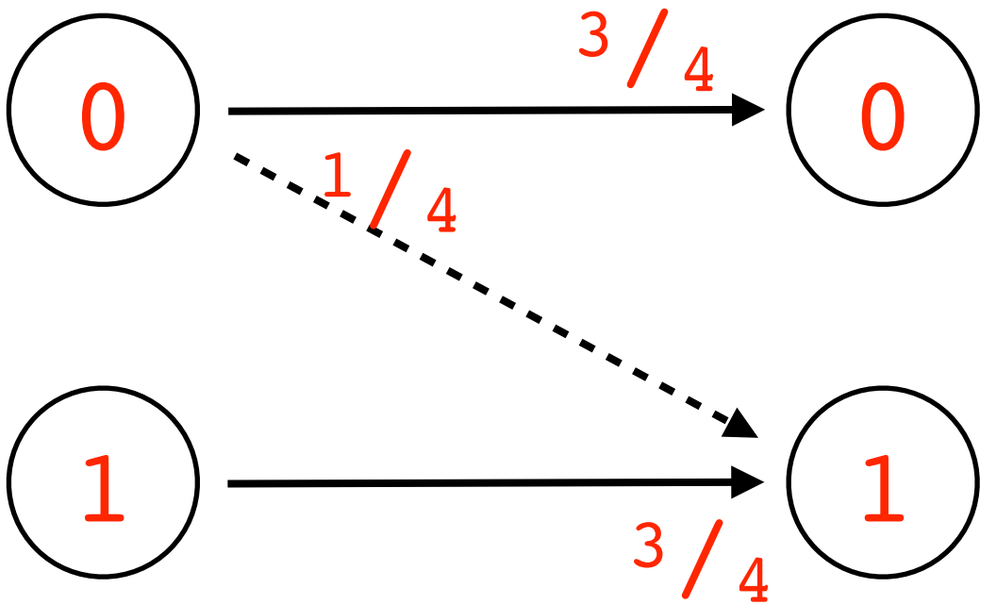
Example: BSC



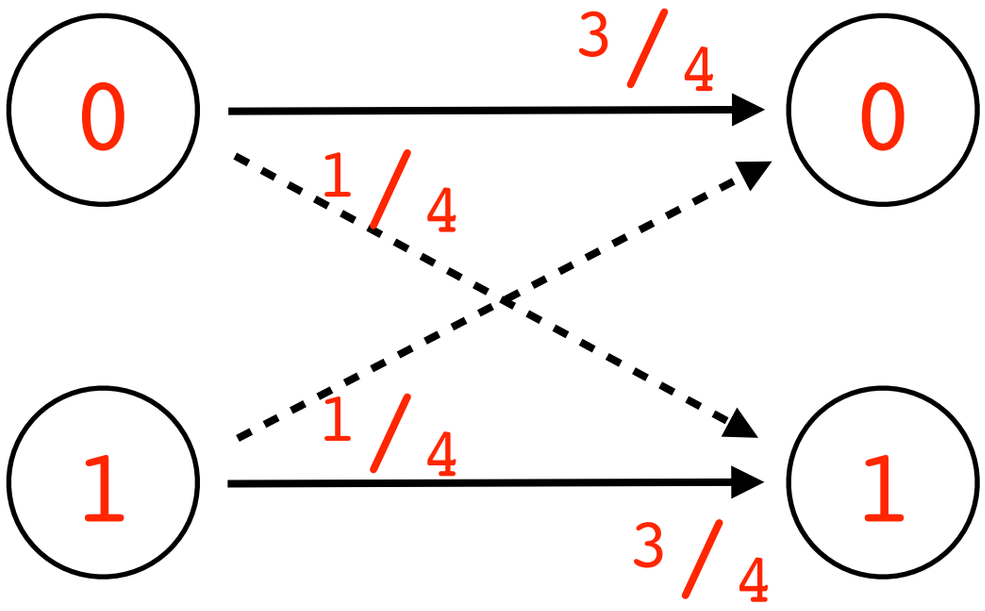
Example: BSC



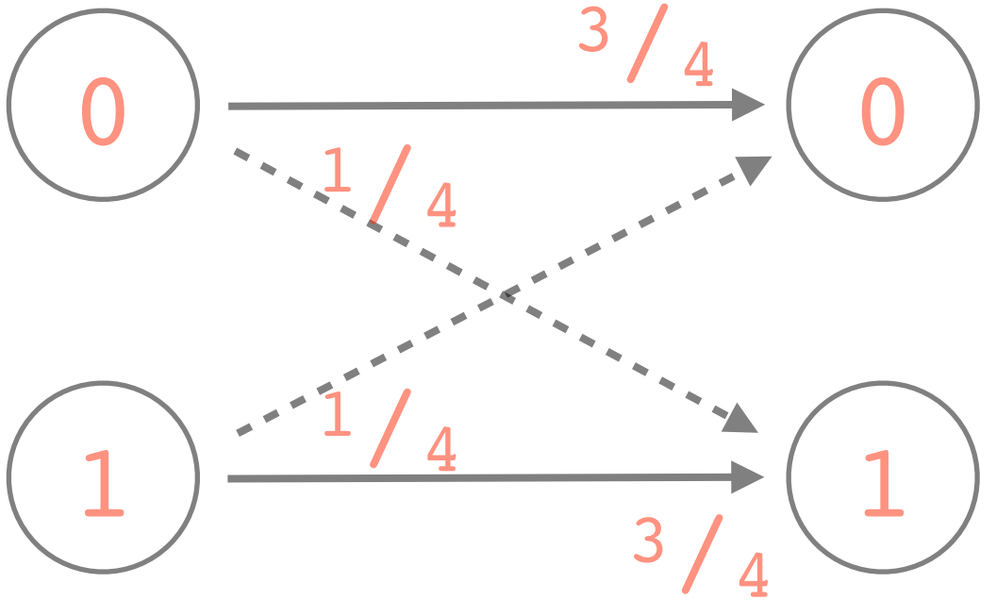
Example: BSC



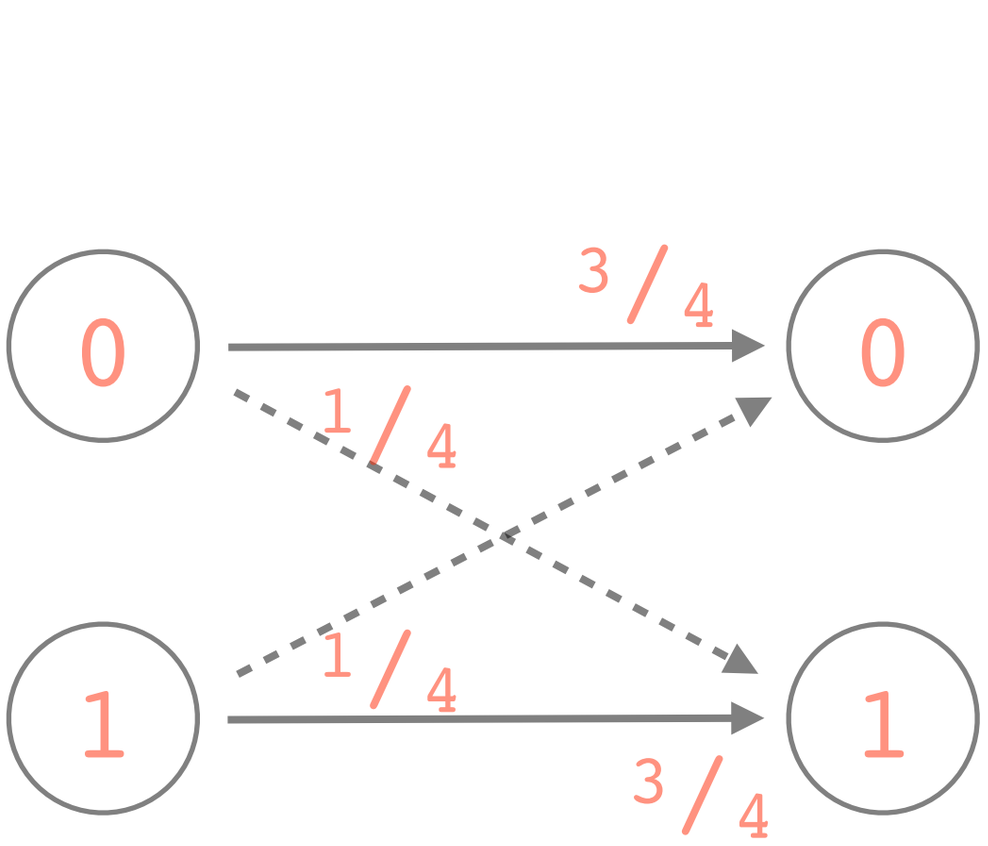
Example: BSC



Example: BSC



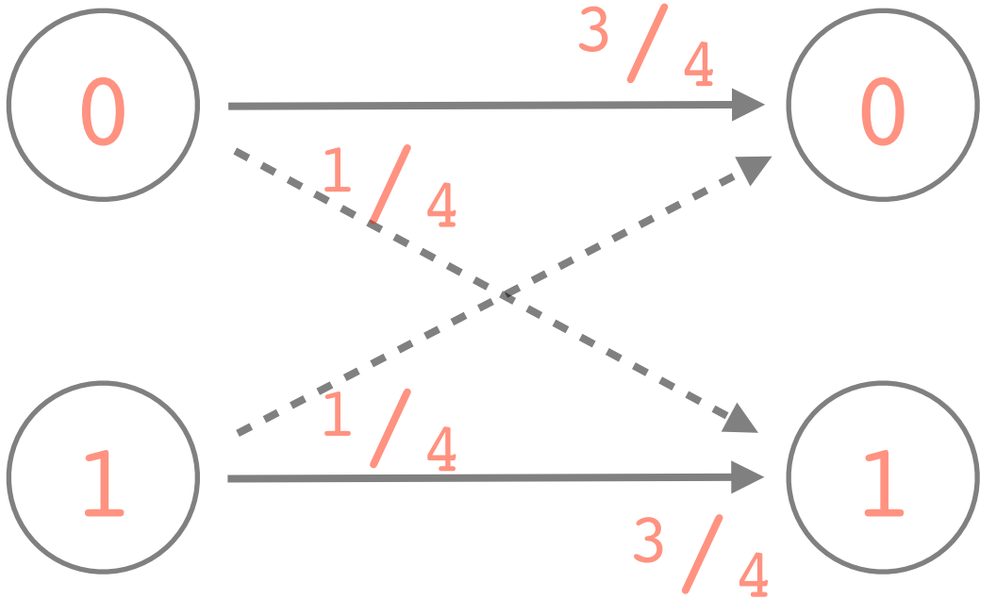
Example: BSC



(0,⊥)

(1,⊥)

Example: BSC



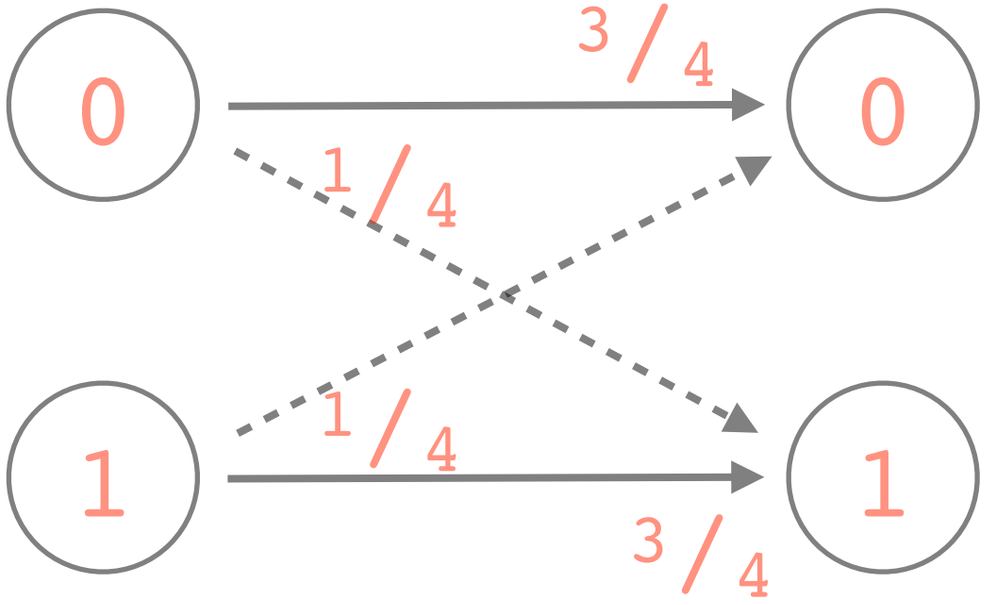
(0,⊥)

(⊥,0)

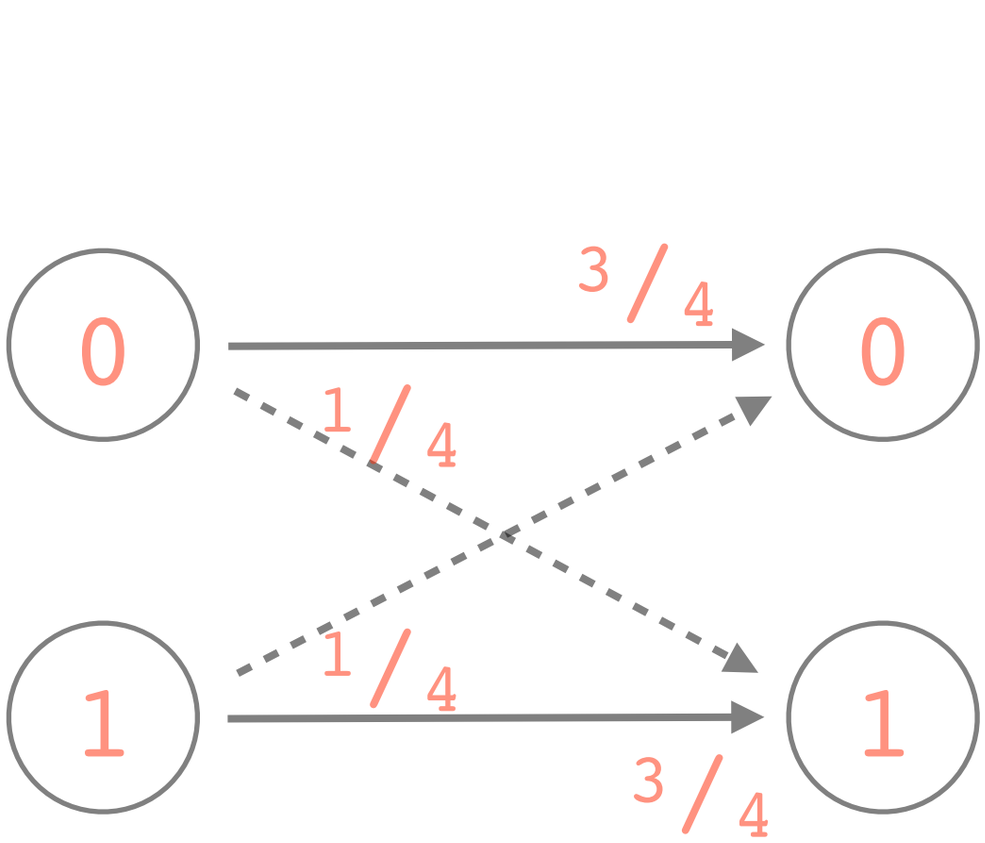
(1,⊥)

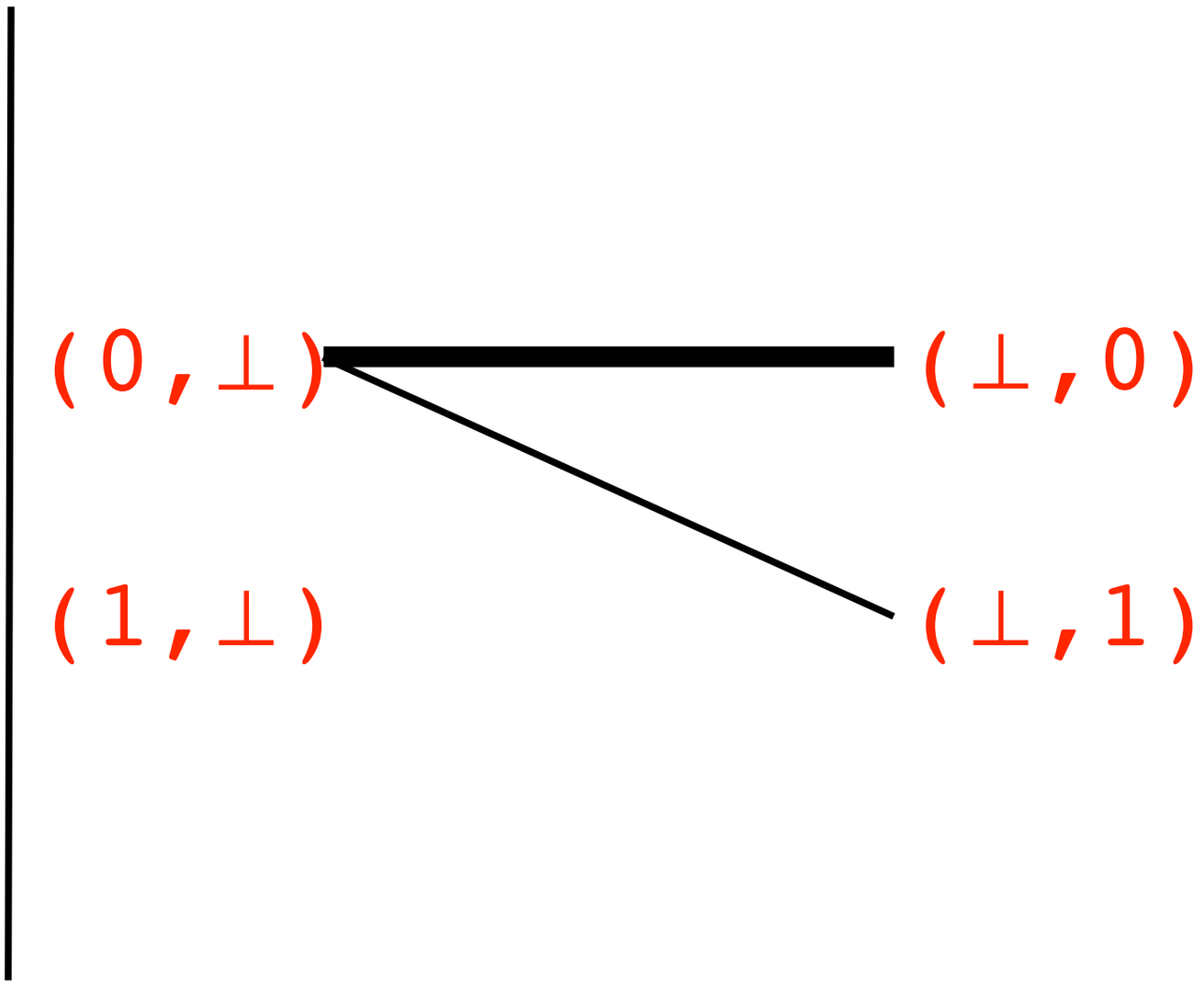
(⊥,1)

Example: BSC

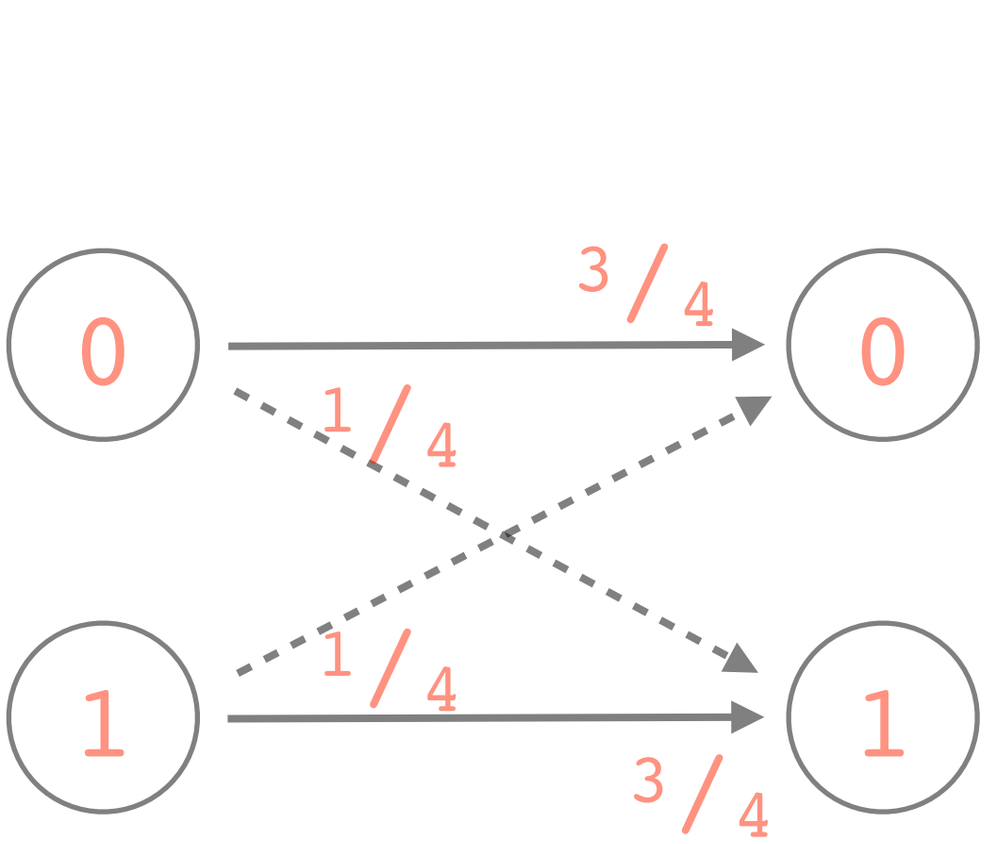


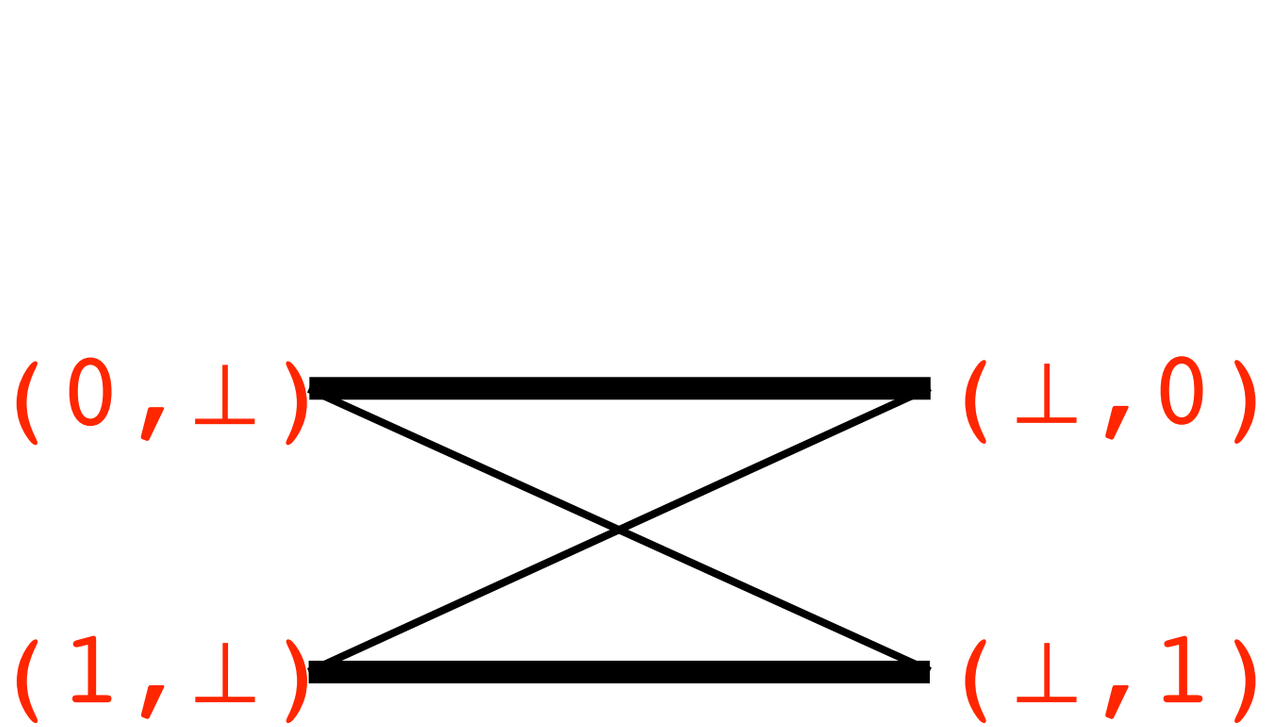
Example: BSC



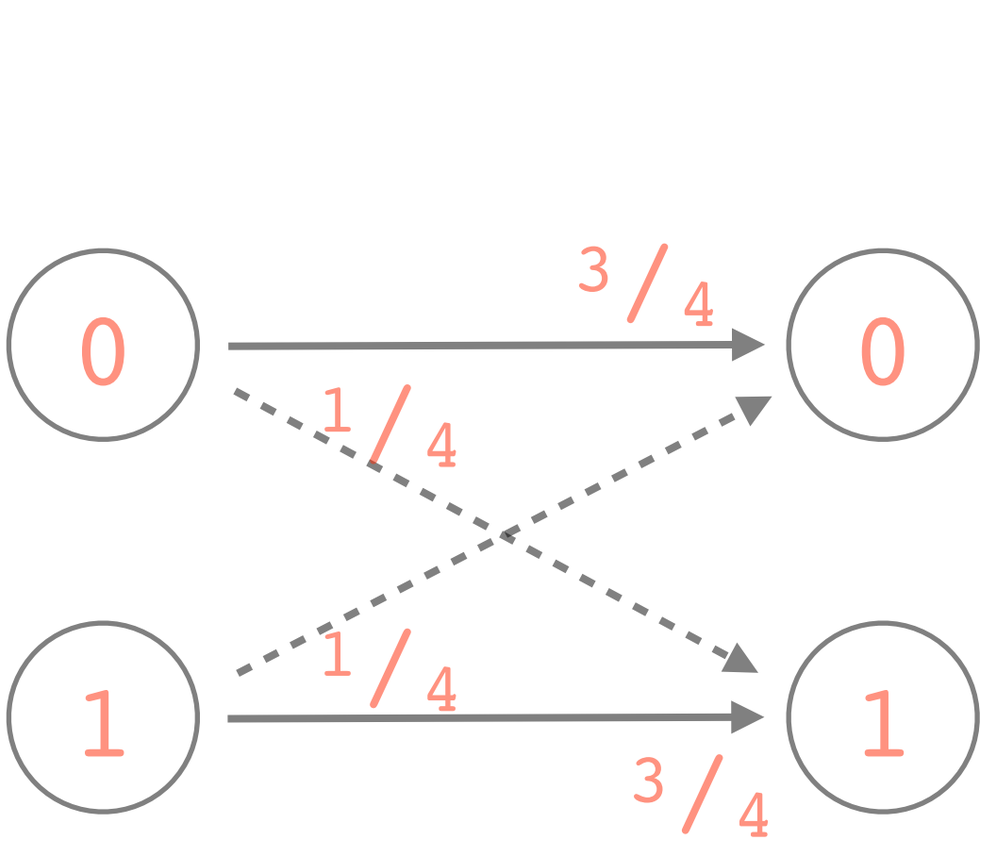


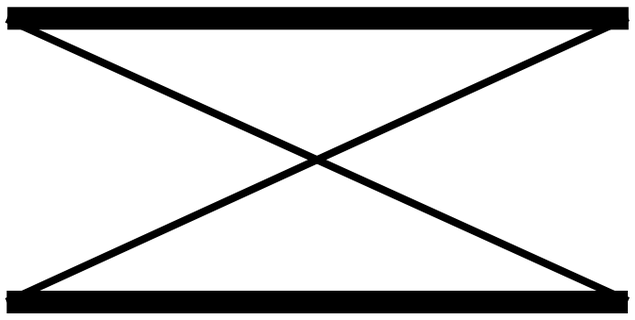
Example: BSC





Example: BSC





Graph of a function

Graph of a function

- A weighted bipartite graph

Graph of a function

- A weighted bipartite graph
- Partite sets are Alice and Bob views

Graph of a function

- A weighted bipartite graph

- Partite sets are Alice and Bob views

- The edge connecting `(x,w)` to `(y,z)` has weight `Pr[ w,z | x,y]`

Product Graphs

Product Graphs

- If a graph has more than one connected component, then a graph is "product graph" if and only if each of its components is a "product graph"

Product Graphs

- If a graph has more than one connected component, then a graph is "product graph" if and only if each of its components is a "product graph"
- A connected component is product graph if:

Product Graphs

- If a graph has more than one connected component, then a graph is "product graph" if and only if each of its components is a "product graph"
- A connected component is product graph if:
  - The conditional distribution on Bob views given any Alice view in the connected component is identical

Product Graphs

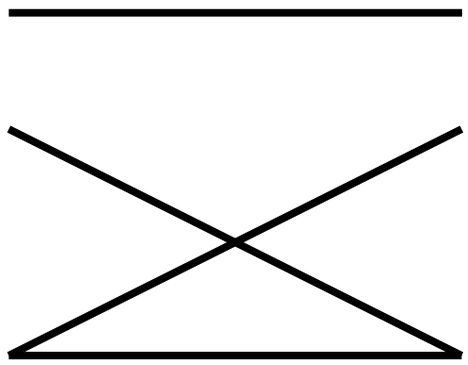
- If a graph has more than one connected component, then a graph is "product graph" if and only if each of its components is a "product graph"
- A connected component is product graph if:
  - The conditional distribution on Bob views given any Alice view in the connected component is identical
  - Note: This is defined using uniform distribution over Bob's inputs

Product Graph Examples

Product Graph Examples

Symmetric OR

Product Graph Examples

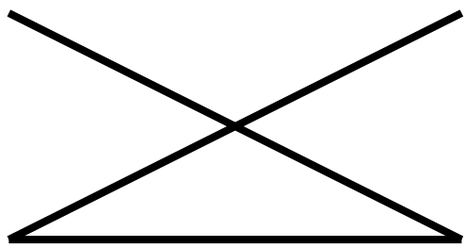


Symmetric OR

Product Graph Examples

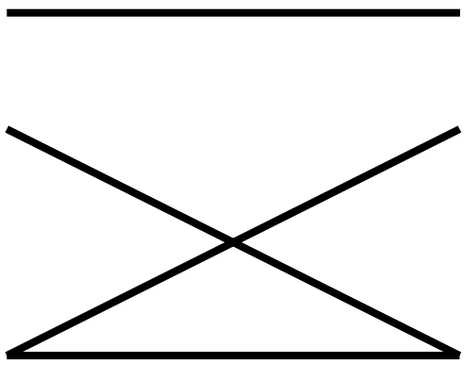
Symmetric OR

Product Graph Examples



Symmetric OR

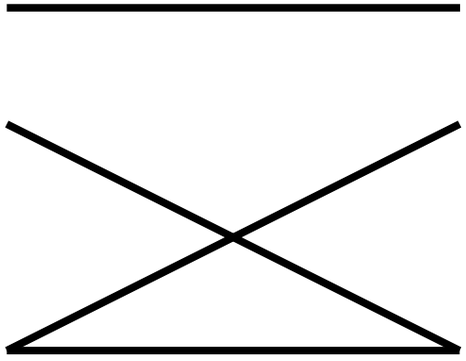
Product Graph Examples



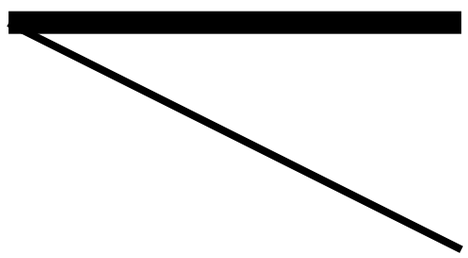
Symmetric OR

BSC

Product Graph Examples

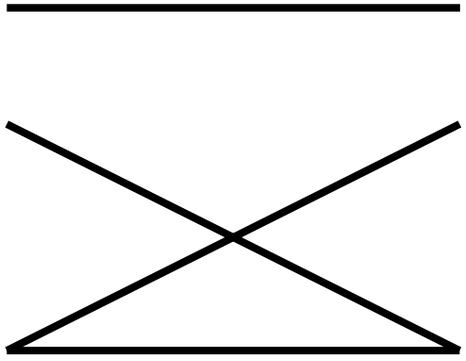


Symmetric OR

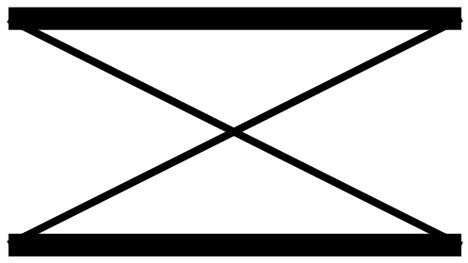


BSC

Product Graph Examples

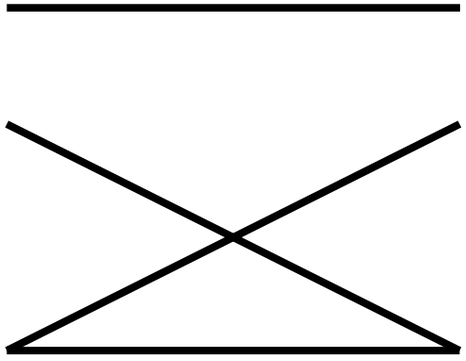


Symmetric OR

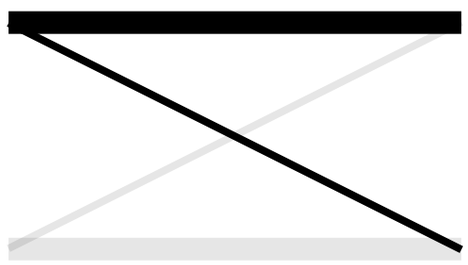


BSC

Product Graph Examples

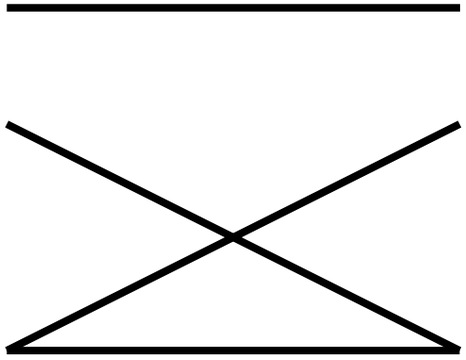


Symmetric OR

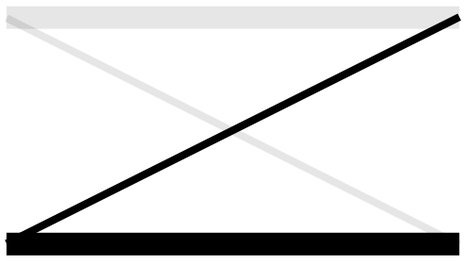


BSC

Product Graph Examples

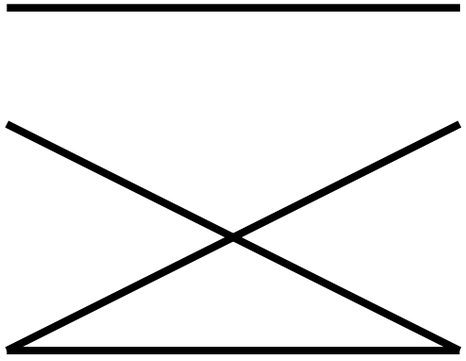


Symmetric OR

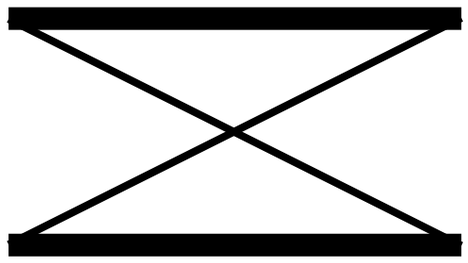


BSC

Product Graph Examples



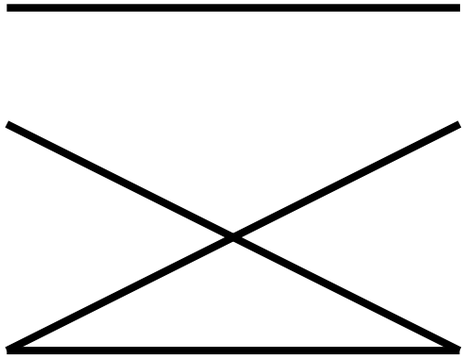
Symmetric OR



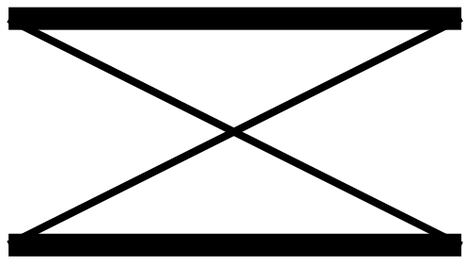
BSC

Inattentive channel

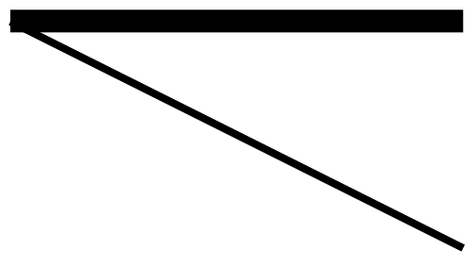
Product Graph Examples



Symmetric OR

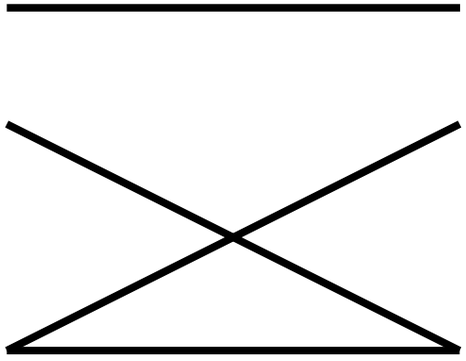


BSC

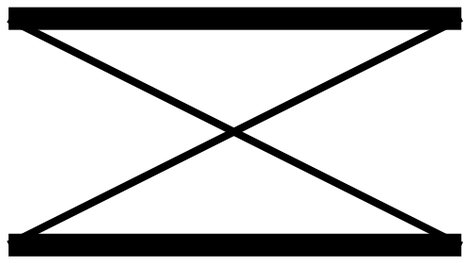


Inattentive channel

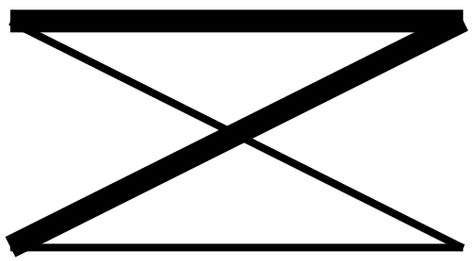
Product Graph Examples



Symmetric OR

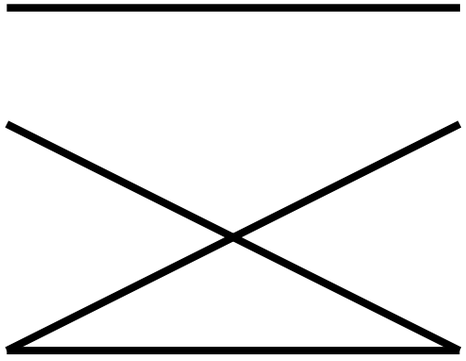


BSC

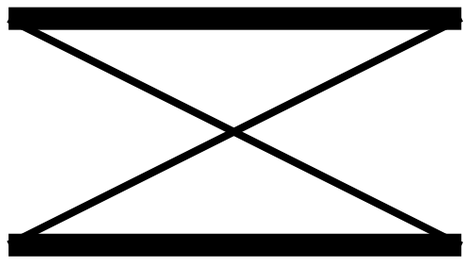


Inattentive channel

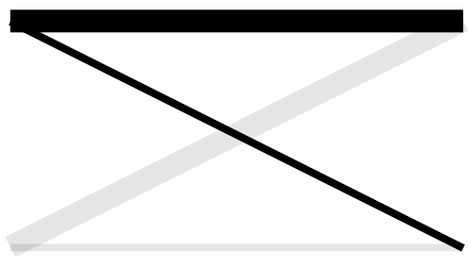
Product Graph Examples



Symmetric OR

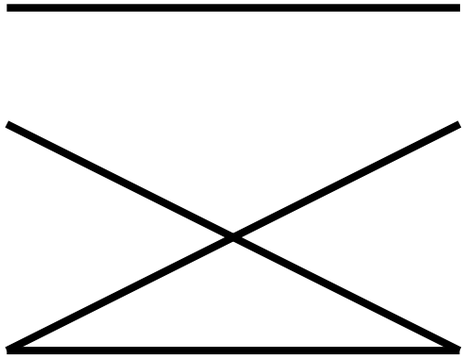


BSC

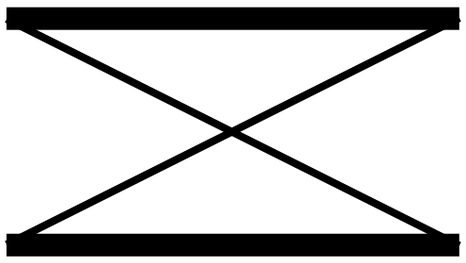


Inattentive channel

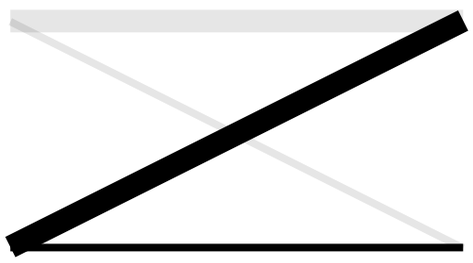
Product Graph Examples



Symmetric OR

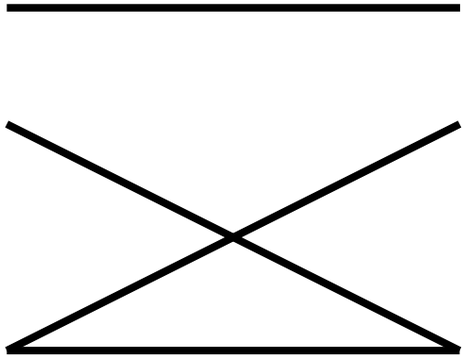


BSC

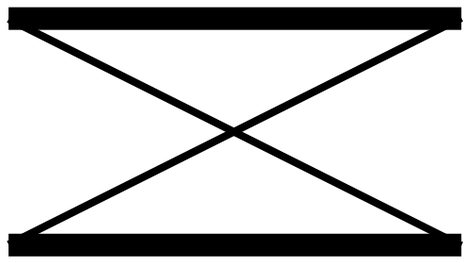


Inattentive channel

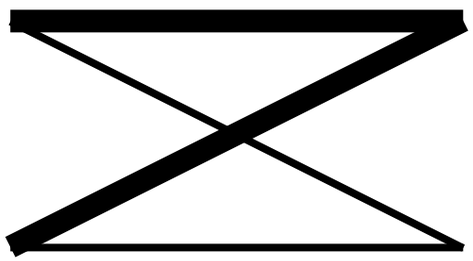
Product Graph Examples



Symmetric OR

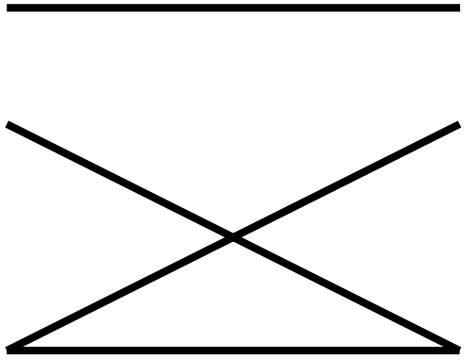


BSC

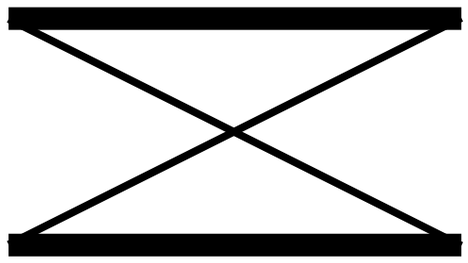


Inattentive channel

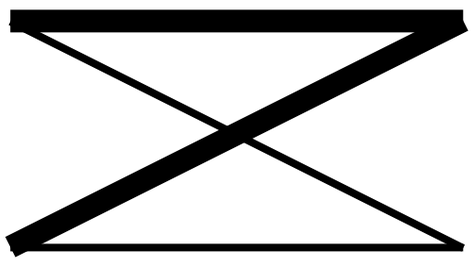
Product Graph Examples



Symmetric OR

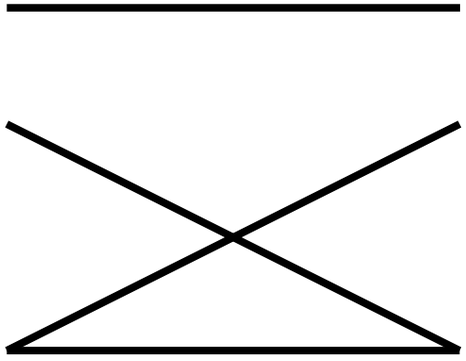


BSC

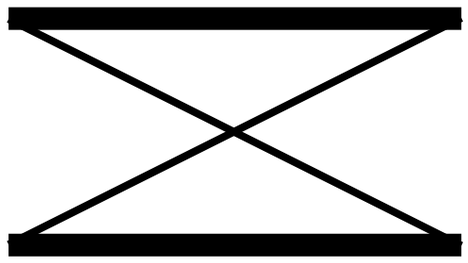


Inattentive channel

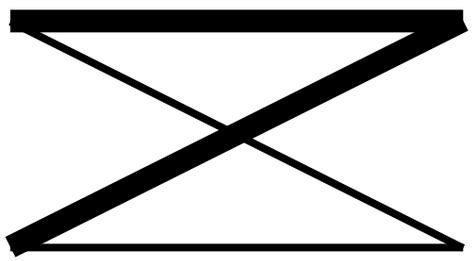
Product Graph Examples



Symmetric OR



BSC



Inattentive channel

Recall the Result

Recall the Result

- `g` is (semi-honest) complete if and only if `graph(g)` is not a product graph

Recall the Result

- `g` is (semi-honest) complete if and only if `graph(g)` is not a product graph
  - Our result only uses [KILIAN-00]'s semi-honest completeness characterization for randomized symmetric SFE

Recall the Result

- `g` is (semi-honest) complete if and only if `graph(g)` is not a product graph
  - Our result only uses [KILIAN-00]'s semi-honest completeness characterization for randomized symmetric SFE
  - This result resolves several open problems in semi-honest completeness characterization!

Malicious Case: Intro

Active Completeness

Active Completeness

- Redundancies: Remove inputs which shall never be used by any (intelligent) malicious adversary

Active Completeness

- Redundancies: Remove inputs which shall never be used by any (intelligent) malicious adversary
- Core of a function: Whatever is left after iterative removal of redundant inputs

Active Completeness

- Redundancies: Remove inputs which shall never be used by any (intelligent) malicious adversary
- Core of a function: Whatever is left after iterative removal of redundant inputs
- A Conjecture made in this paper: `g` is malicious complete if and only if `graph(core(g))` is not a product graph

Active Completeness

- Redundancies: Remove inputs which shall never be used by any (intelligent) malicious adversary
- Core of a function: Whatever is left after iterative removal of redundant inputs
- A Conjecture made in this paper: `g` is malicious complete if and only if `graph(core(g))` is not a product graph
  - Recently proven by [Maji-Prabhakaran-12]

Open Problems

Open Problems

Open Problems

- Multi-party

Open Problems

- Multi-party
  - Three or more parties

Open Problems

- Multi-party
  - Three or more parties
  - Point-to-point channels and Broadcast channels

Open Problems

- Multi-party
  - Three or more parties
  - Point-to-point channels and Broadcast channels
- Interactive functionalities

Open Problems

- Multi-party
  - Three or more parties
  - Point-to-point channels and Broadcast channels
- Interactive functionalities
  - Modeled as finite alphabet transducers

Open Problems

- Multi-party
  - Three or more parties
  - Point-to-point channels and Broadcast channels
- Interactive functionalities
  - Modeled as finite alphabet transducers
- Rates

Open Problems

- Multi-party
  - Three or more parties
  - Point-to-point channels and Broadcast channels
- Interactive functionalities
  - Modeled as finite alphabet transducers
- Rates
  - How efficiently can copies of g be securely morphed into copies of f

Thanks