Exercise 3: Complexity of First-Order Queries

Database Theory

2020-04-27

Maximilian Marx, David Carral

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\}$$

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\}$$

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \text{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \text{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- Note that a BCQ q is entailed in $\mathcal{I}$ iff $M[q](\mathcal{I}) \neq \emptyset$. Thus, a TM deciding QE also decides BQE.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \text{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- Note that a BCQ q is entailed in $\mathcal{I}$ iff $M[q](\mathcal{I}) \neq \emptyset$. Thus, a TM deciding QE also decides BQE.
- We show that using a TM deciding BQE, we can construct a TM deciding QA, and

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \text{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- Note that a BCQ q is entailed in $\mathcal{I}$ iff $M[q](\mathcal{I}) \neq \emptyset$. Thus, a TM deciding QE also decides BQE.
- We show that using a TM deciding BQE, we can construct a TM deciding QA, and
- that using a TM deciding QA we can construct a TM deciding QE.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \text{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA:

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \text{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA:
- Let $\mathcal{M}$ be a TM deciding BQE.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \text{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA:
- Let $\mathcal{M}$ be a TM deciding BQE.
- Construct the TM $\mathcal{M}'$ that, on input $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle$ with $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{c} = \langle c_1, \ldots, c_n \rangle$:

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\mathrm{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \mathrm{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \mathrm{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA:
- Let $\mathcal{M}$ be a TM deciding BQE.
- Construct the TM $\mathcal{M}'$ that, on input $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle$ with $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{c} = \langle c_1, \ldots, c_n \rangle$:
  1. transforms $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle$ into $\langle \mathcal{I}, q[x_1/c_1, \ldots, x_n/c_n] \rangle$,

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\mathrm{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \mathrm{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \mathrm{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA:
- Let $\mathcal{M}$ be a TM deciding BQE.
- Construct the TM $\mathcal{M}'$ that, on input $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle$ with $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{c} = \langle c_1, \ldots, c_n \rangle$:
    1. transforms $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle$ into $\langle \mathcal{I}, q[x_1/c_1, \ldots, x_n/c_n] \rangle$,
    2. simulates $\mathcal{M}$ on input $\langle \mathcal{I}, q[x_1/c_1, \ldots, x_n/c_n] \rangle$, and

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\mathrm{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \mathrm{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \mathrm{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA:
- Let $\mathcal{M}$ be a TM deciding BQE.
- Construct the TM $\mathcal{M}'$ that, on input $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle$ with $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{c} = \langle c_1, \ldots, c_n \rangle$:
  1. transforms $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle$ into $\langle \mathcal{I}, q[x_1/c_1, \ldots, x_n/c_n] \rangle$,
  2. simulates $\mathcal{M}$ on input $\langle \mathcal{I}, q[x_1/c_1, \ldots, x_n/c_n] \rangle$, and
  3. accepts iff $\mathcal{M}$ accepts.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \text{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA:
- Let $\mathcal{M}$ be a TM deciding BQE.
- Construct the TM $\mathcal{M}'$ that, on input $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle$ with $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{c} = \langle c_1, \ldots, c_n \rangle$:
  1. transforms $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle$ into $\langle \mathcal{I}, q[x_1/c_1, \ldots, x_n/c_n] \rangle$,
  2. simulates $\mathcal{M}$ on input $\langle \mathcal{I}, q[x_1/c_1, \ldots, x_n/c_n] \rangle$, and
  3. accepts iff $\mathcal{M}$ accepts.
- Then $\mathcal{M}'$ decides QA.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \text{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA, and
- that using a TM deciding QA we can construct a TM deciding QE:
- Let $\mathcal{M}$ be a TM deciding QA.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\mathrm{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \mathrm{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \mathrm{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA, and
- that using a TM deciding QA we can construct a TM deciding QE:
- Let $\mathcal{M}$ be a TM deciding QA.
- Construct the TM $\mathcal{M}'$ that, on input $\langle \mathcal{I}, q[\mathbf{x}] \rangle$ with $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$:

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \text{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \text{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA, and
- that using a TM deciding QA we can construct a TM deciding QE:
- Let $\mathcal{M}$ be a TM deciding QA.
- Construct the TM $\mathcal{M}'$ that, on input $\langle \mathcal{I}, q[\mathbf{x}] \rangle$ with $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$:
  1. If $n = 0$, then $\mathcal{M}'$ simulates $\mathcal{M}$ on input $\langle \mathcal{I}, q, \langle\rangle \rangle$ and accept iff the simulation accepts.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\mathrm{BQE} = \left\{ \langle \mathcal{I}, q \rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q \right\} \quad \mathrm{QA} = \left\{ \langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I}) \right\} \quad \mathrm{QE} = \left\{ \langle \mathcal{I}, q[\mathbf{x}] \rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset \right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA, and
- that using a TM deciding QA we can construct a TM deciding QE:
- Let $\mathcal{M}$ be a TM deciding QA.
- Construct the TM $\mathcal{M}'$ that, on input $\langle \mathcal{I}, q[\mathbf{x}] \rangle$ with $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$:
  1. If $n = 0$, then $\mathcal{M}'$ simulates $\mathcal{M}$ on input $\langle \mathcal{I}, q, \langle\rangle \rangle$ and accept iff the simulation accepts.
  2. Otherwise, $\mathcal{M}'$ simulates $\mathcal{M}$ on all inputs $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c} \rangle$ with $\mathbf{c} \in \mathbf{adom}(\mathcal{I}, q)^n$ and accepts if any simulation accepts.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.
Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{\langle \mathcal{I}, q\rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q\right\} \quad \text{QA} = \left\{\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c}\rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I})\right\} \quad \text{QE} = \left\{\langle \mathcal{I}, q[\mathbf{x}]\rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset\right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA, and
- that using a TM deciding QA we can construct a TM deciding QE:
- Let $\mathcal{M}$ be a TM deciding QA.
- Construct the TM $\mathcal{M}'$ that, on input $\langle \mathcal{I}, q[\mathbf{x}]\rangle$ with $\mathbf{x} = \langle x_1, \ldots, x_n\rangle$:
  1. If $n = 0$, then $\mathcal{M}'$ simulates $\mathcal{M}$ on input $\langle \mathcal{I}, q, \langle\rangle\rangle$ and accept iff the simulation accepts.
  2. Otherwise, $\mathcal{M}'$ simulates $\mathcal{M}$ on all inputs $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c}\rangle$ with $\mathbf{c} \in \mathbf{adom}(\mathcal{I}, q)^n$ and accepts if any simulation accepts.
  3. If no simulation accepts, $\mathcal{M}'$ rejects.

Exercise 1

Exercise. We consider three problems related to query answering in the lecture:

Boolean Query Entailment Given a Boolean query q and a database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold?

Query Answering Given an n-ary query q, a database instance $\mathcal{I}$, and an n-ary tuple $\mathbf{c}$, does $\mathbf{c} \in M[q](\mathcal{I})$ hold?

Query Emptiness Given a query q and a database instance $\mathcal{I}$, is $M[q](\mathcal{I}) \neq \emptyset$?

Show that these problems are equivalent, i.e., show that any algorithm solving one of these problems, it can also be used to solve the others.

Solution.

- We restate the problems as decision problems:

$$\text{BQE} = \left\{\langle \mathcal{I}, q\rangle \,\middle|\, q \text{ a BCQ with } \mathcal{I} \models q\right\} \quad \text{QA} = \left\{\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c}\rangle \,\middle|\, \mathbf{c} \in M[q](\mathcal{I})\right\} \quad \text{QE} = \left\{\langle \mathcal{I}, q[\mathbf{x}]\rangle \,\middle|\, M[q](\mathcal{I}) \neq \emptyset\right\}$$

- We show that using a TM deciding BQE, we can construct a TM deciding QA, and
- that using a TM deciding QA we can construct a TM deciding QE:
- Let $\mathcal{M}$ be a TM deciding QA.
- Construct the TM $\mathcal{M}'$ that, on input $\langle \mathcal{I}, q[\mathbf{x}]\rangle$ with $\mathbf{x} = \langle x_1, \ldots, x_n\rangle$:
  1. If $n = 0$, then $\mathcal{M}'$ simulates $\mathcal{M}$ on input $\langle \mathcal{I}, q, \langle\rangle\rangle$ and accept iff the simulation accepts.
  2. Otherwise, $\mathcal{M}'$ simulates $\mathcal{M}$ on all inputs $\langle \mathcal{I}, q[\mathbf{x}], \mathbf{c}\rangle$ with $\mathbf{c} \in \textbf{adom}(\mathcal{I}, q)^n$ and accepts if any simulation accepts.
  3. If no simulation accepts, $\mathcal{M}'$ rejects.
- Then $\mathcal{M}'$ decides QE.

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LOGSPACE transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LOGSPACE transducer is the contents of its output tape when it halts, i.e., LOGSPACE transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$:

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i=a_j}(R)$:
- 1. We use the unnamed perspective, encoding attributes a_i and a_j as numbers i and j, and storing the table R as a sequence of rows of the form $\$c_1, \ldots, c_n\#$.

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$:
-
  1. We use the unnamed perspective, encoding attributes a_i and a_j as numbers i and j, and storing the table R as a sequence of rows of the form $\$c_1, \ldots, c_n\#$.
  2. We use three pointers p_r, p_i, and p_j.

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$:
- 1. We use the unnamed perspective, encoding attributes a_i and a_j as numbers i and j, and storing the table R as a sequence of rows of the form $\$c_1, \ldots, c_n\#$.
  2. We use three pointers p_r, p_i, and p_j.
  3. Initially, p_r points to the first \$ symbol, and we repeat:

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LOGSPACE transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LOGSPACE transducer is the contents of its output tape when it halts, i.e., LOGSPACE transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LOGSPACE transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$:
-
  1. We use the unnamed perspective, encoding attributes a_i and a_j as numbers i and j, and storing the table R as a sequence of rows of the form $\$c_1, \ldots, c_n\#$.
  2. We use three pointers p_r, p_i, and p_j.
  3. Initially, p_r points to the first \$ symbol, and we repeat:
     3.1 point p_i at the beginning of the i-th constant of the row;

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$:
- 1. We use the unnamed perspective, encoding attributes a_i and a_j as numbers i and j, and storing the table R as a sequence of rows of the form $\$c_1, \ldots, c_n\#$.
  2. We use three pointers p_r, p_i, and p_j.
  3. Initially, p_r points to the first \$ symbol, and we repeat:
     - 3.1 point p_i at the beginning of the i-th constant of the row;
     - 3.2 point p_j at the beginning of the j-th constant of the row;

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LOGSPACE transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LOGSPACE transducer is the contents of its output tape when it halts, i.e., LOGSPACE transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LOGSPACE transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$:
- 1. We use the unnamed perspective, encoding attributes a_i and a_j as numbers i and j, and storing the table R as a sequence of rows of the form $\$c_1, \ldots, c_n\#$.
  2. We use three pointers p_r, p_i, and p_j.
  3. Initially, p_r points to the first \$ symbol, and we repeat:
     - 3.1 point p_i at the beginning of the i-th constant of the row;
     - 3.2 point p_j at the beginning of the j-th constant of the row;
     - 3.3 using p_i and p_j compare the two constants.

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$:
- 1. We use the unnamed perspective, encoding attributes a_i and a_j as numbers i and j, and storing the table R as a sequence of rows of the form $\$c_1, \ldots, c_n\#$.
  2. We use three pointers p_r, p_i, and p_j.
  3. Initially, p_r points to the first \$ symbol, and we repeat:
     - 3.1 point p_i at the beginning of the i-th constant of the row;
     - 3.2 point p_j at the beginning of the j-th constant of the row;
     - 3.3 using p_i and p_j compare the two constants.
     - 3.4 if the constants are equal, copy the row to the output tape (using p_r); and

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$:
- 1. We use the unnamed perspective, encoding attributes a_i and a_j as numbers i and j, and storing the table R as a sequence of rows of the form $\$c_1, \ldots, c_n\#$.
  2. We use three pointers p_r, p_i, and p_j.
  3. Initially, p_r points to the first \$ symbol, and we repeat:
     - 3.1 point p_i at the beginning of the i-th constant of the row;
     - 3.2 point p_j at the beginning of the j-th constant of the row;
     - 3.3 using p_i and p_j compare the two constants.
     - 3.4 if the constants are equal, copy the row to the output tape (using p_r); and
     - 3.5 point p_r to the next \$, if there is any, otherwise halt.

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$.
- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $\{ a'_1, \ldots, a'_\ell \} \subseteq \{ a_1, \ldots, a_n \}$, computes $\pi_{a'_1, \ldots, a'_\ell}(R)$:

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$.
- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $\{ a'_1, \ldots, a'_\ell \} \subseteq \{ a_1, \ldots, a_n \}$, computes $\pi_{a'_1, \ldots, a'_\ell}(R)$:
- 1. We use the named perspective, encoding the set of attributes $\{ a'_1, \ldots, a'_n \}$ as $\#a'_1, \ldots, a'_n\#$ at the start of the input, and then encoding R as $\$a_1 \mapsto c^i_1, \ldots, a_n \mapsto c^i_n\$$.
  2. We point a pointer p_c to the first attribute a'_1, and, for every row of the input, proceed:

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \dots, a_n]$ and some $a_i, a_j \in \{ a_1, \dots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$.
- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \dots, a_n]$ and some $\{ a'_1, \dots, a'_\ell \} \subseteq \{ a_1, \dots, a_n \}$, computes $\pi_{a'_1, \dots, a'_\ell}(R)$:
- 1. We use the named perspective, encoding the set of attributes $\{ a'_1, \dots, a'_n \}$ as $\# a'_1, \dots, a'_n \#$ at the start of the input, and then encoding R as $\$ a_1 \mapsto c^i_1, \dots, a_n \mapsto c^i_n \$$.
  2. We point a pointer p_c to the first attribute a'_1, and, for every row of the input, proceed:
     2.1 write \$ to the output.

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$.
- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $\{ a'_1, \ldots, a'_\ell \} \subseteq \{ a_1, \ldots, a_n \}$, computes $\pi_{a'_1, \ldots, a'_\ell}(R)$:
-
  1. We use the named perspective, encoding the set of attributes $\{ a'_1, \ldots, a'_n \}$ as $\#a'_1, \ldots, a'_n\#$ at the start of the input, and then encoding R as $\$a_1 \mapsto c^i_1, \ldots, a_n \mapsto c^i_n\$$.
  2. We point a pointer p_c to the first attribute a'_1, and, for every row of the input, proceed:
     - 2.1 write \$ to the output.
     - 2.2 for every pair $a_j \mapsto c^i_j$, check whether a_j occurs in $\{ a'_1, \ldots, a'_n \}$ and write $a_j \mapsto c^i_j$ if that is the case.

Exercise 2

Exercise. It was shown in the lecture that joins can be computed in logarithmic space. Outline algorithms that implement *selection*, and *projection* in logarithmic space.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $a_i, a_j \in \{ a_1, \ldots, a_n \}$, computes $\sigma_{a_i = a_j}(R)$.
- We describe a LogSpace transducer $\mathcal{M}$ that, given a table R with schema $R[a_1, \ldots, a_n]$ and some $\{ a'_1, \ldots, a'_\ell \} \subseteq \{ a_1, \ldots, a_n \}$, computes $\pi_{a'_1, \ldots, a'_\ell}(R)$:
- 1. We use the named perspective, encoding the set of attributes $\{ a'_1, \ldots, a'_n \}$ as $\# a'_1, \ldots, a'_n \#$ at the start of the input, and then encoding R as $\$ a_1 \mapsto c^i_1, \ldots, a_n \mapsto c^i_n \$$.
  2. We point a pointer p_c to the first attribute a'_1, and, for every row of the input, proceed:
     - 2.1 write \$ to the output.
     - 2.2 for every pair $a_j \mapsto c^i_j$, check whether a_j occurs in $\{ a'_1, \ldots, a'_n \}$ and write $a_j \mapsto c^i_j$ if that is the case.
     - 2.3 write \$ to the output.

Exercise 3

Exercise. Expressions of relational algebra under named perspective can be translated into Boolean circuits, in a similar fashion to the translation illustrated for FO queries in the lecture. Show how each operator of relational algebra gives rise to a corresponding circuit by describing the circuits for the following expressions:

$\sigma_{i=c}(R)$ (c a constant)	$\sigma_{i=j}(R)$ (j an attribute)
$\pi_{a_1,\dots,a_\ell}(R)$	$R \bowtie S$
$\delta_{a_1,\dots,a_\ell \to b_1,\dots,b_\ell}(R)$	$R - S$
$R \cup S$	$R \cap S$

Exercise 3

Exercise. Expressions of relational algebra under named perspective can be translated into Boolean circuits, in a similar fashion to the translation illustrated for FO queries in the lecture. Show how each operator of relational algebra gives rise to a corresponding circuit by describing the circuits for the following expressions:

$\sigma_{i=c}(R)$ (c a constant)	$\sigma_{i=j}(R)$ (j an attribute)
$\pi_{a_1,\ldots,a_\ell}(R)$	$R \bowtie S$
$\delta_{a_1,\ldots,a_\ell \to b_1,\ldots,b_\ell}(R)$	$R - S$
$R \cup S$	$R \cap S$

Solution.

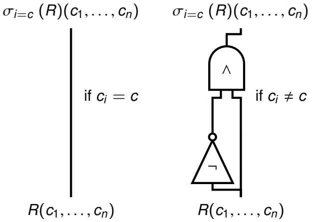
Exercise 3

Exercise. Expressions of relational algebra under named perspective can be translated into Boolean circuits, in a similar fashion to the translation illustrated for FO queries in the lecture. Show how each operator of relational algebra gives rise to a corresponding circuit by describing the circuits for the following expressions:

$\sigma_{i=c}(R)$ (c a constant)	$\sigma_{i=j}(R)$ (j an attribute)
$\pi_{a_1,\ldots,a_\ell}(R)$	$R \bowtie S$
$\delta_{a_1,\ldots,a_\ell \to b_1,\ldots,b_\ell}(R)$	$R - S$
$R \cup S$	$R \cap S$

Solution.

$\sigma_{i=c}(R)$ for each tuple $\langle c_1, \ldots, c_n \rangle$ in R, we add one of these two circuits:



Exercise 3

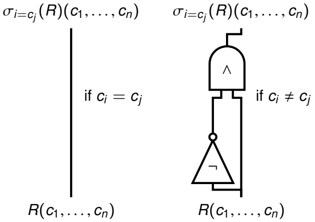
Exercise. Expressions of relational algebra under named perspective can be translated into Boolean circuits, in a similar fashion to the translation illustrated for FO queries in the lecture. Show how each operator of relational algebra gives rise to a corresponding circuit by describing the circuits for the following expressions:

$\sigma_{i=c}(R)$ (c a constant)	$\sigma_{i=j}(R)$ (j an attribute)
$\pi_{a_1,\ldots,a_\ell}(R)$	$R \bowtie S$
$\delta_{a_1,\ldots,a_\ell \to b_1,\ldots,b_\ell}(R)$	$R - S$
$R \cup S$	$R \cap S$

Solution.

$\sigma_{i=c}(R)$ for each tuple $\langle c_1, \ldots, c_n \rangle$ in R, we add one of these two circuits:

$\sigma_{i=j}(R)$ analogous.



Exercise 3

Exercise. Expressions of relational algebra under named perspective can be translated into Boolean circuits, in a similar fashion to the translation illustrated for FO queries in the lecture. Show how each operator of relational algebra gives rise to a corresponding circuit by describing the circuits for the following expressions:

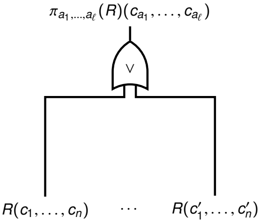
$\sigma_{i=c}(R)$ (c a constant)	$\sigma_{i=j}(R)$ (j an attribute)
$\pi_{a_1,\dots,a_\ell}(R)$	$R \bowtie S$
$\delta_{a_1,\dots,a_\ell \to b_1,\dots,b_\ell}(R)$	$R - S$
$R \cup S$	$R \cap S$

Solution.

$\sigma_{i=c}(R)$ for each tuple $\langle c_1, \dots, c_n \rangle$ in R, we add one of these two circuits:

$\sigma_{i=j}(R)$ analogous.

$\pi_{a_1,\dots,a_\ell}(R)$ for all tuples $\langle c_1, \dots, c_n \rangle, \dots, \langle c'_1, \dots, c'_n \rangle$ in R with $c_{a_1} = c'_{a_1}, \dots, c_{a_\ell} = c'_{a_\ell}$, we add the circuit:



Exercise 3

Exercise. Expressions of relational algebra under named perspective can be translated into Boolean circuits, in a similar fashion to the translation illustrated for FO queries in the lecture. Show how each operator of relational algebra gives rise to a corresponding circuit by describing the circuits for the following expressions:

$\sigma_{i=c}(R)$ (c a constant)	$\sigma_{i=j}(R)$ (j an attribute)
$\pi_{a_1,\ldots,a_\ell}(R)$	$R \bowtie S$
$\delta_{a_1,\ldots,a_\ell \to b_1,\ldots,b_\ell}(R)$	$R - S$
$R \cup S$	$R \cap S$

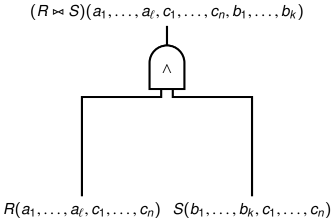
Solution.

$\sigma_{i=c}(R)$ for each tuple $\langle c_1, \ldots, c_n \rangle$ in R, we add one of these two circuits:

$\sigma_{i=j}(R)$ analogous.

$\pi_{a_1,\ldots,a_\ell}(R)$ for all tuples $\langle c_1, \ldots, c_n \rangle, \ldots, \langle c'_1, \ldots, c'_n \rangle$ in R with $c_{a_1} = c'_{a_1}, \ldots, c_{a_\ell} = c'_{a_\ell}$, we add the circuit:

$R \bowtie S$ for each tuple $\langle a_1, \ldots, a_\ell, c_1, \ldots, c_n \rangle$ in R and each tuple $\langle b_1, \ldots, b_k, c_1, \ldots, c_n \rangle$ in S, we add the circuit:



Exercise 3

Exercise. Expressions of relational algebra under named perspective can be translated into Boolean circuits, in a similar fashion to the translation illustrated for FO queries in the lecture. Show how each operator of relational algebra gives rise to a corresponding circuit by describing the circuits for the following expressions:

$\sigma_{i=c}(R)$ (c a constant)	$\sigma_{i=j}(R)$ (j an attribute)
$\pi_{a_1,\ldots,a_\ell}(R)$	$R \bowtie S$
$\delta_{a_1,\ldots,a_\ell \to b_1,\ldots,b_\ell}(R)$	$R - S$
$R \cup S$	$R \cap S$

Solution.

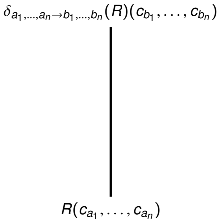
$\sigma_{i=c}(R)$ for each tuple $\langle c_1, \ldots, c_n \rangle$ in R, we add one of these two circuits:

$\sigma_{i=j}(R)$ analogous.

$\pi_{a_1,\ldots,a_\ell}(R)$ for all tuples $\langle c_1, \ldots, c_n \rangle, \ldots, \langle c'_1, \ldots, c'_n \rangle$ in R with $c_{a_1} = c'_{a_1}, \ldots, c_{a_\ell} = c'_{a_\ell}$, we add the circuit:

$R \bowtie S$ for each tuple $\langle a_1, \ldots, a_\ell, c_1, \ldots, c_n \rangle$ in R and each tuple $\langle b_1, \ldots, b_k, c_1, \ldots, c_n \rangle$ in S, we add the circuit:

$\delta_{a_1,\ldots a_n \to b_1,\ldots,b_n}(R)$ for each tuple $\langle c_{a_1}, \ldots, c_{a_n} \rangle$ in R, we add the circuit:



Exercise 3

Exercise. Expressions of relational algebra under named perspective can be translated into Boolean circuits, in a similar fashion to the translation illustrated for FO queries in the lecture. Show how each operator of relational algebra gives rise to a corresponding circuit by describing the circuits for the following expressions:

$\sigma_{i=c}(R)$ (c a constant)	$\sigma_{i=j}(R)$ (j an attribute)
$\pi_{a_1,\ldots,a_\ell}(R)$	$R \bowtie S$
$\delta_{a_1,\ldots,a_\ell \to b_1,\ldots,b_\ell}(R)$	$R - S$
$R \cup S$	$R \cap S$

Solution.

$\sigma_{i=c}(R)$ for each tuple $\langle c_1, \ldots, c_n \rangle$ in R, we add one of these two circuits:

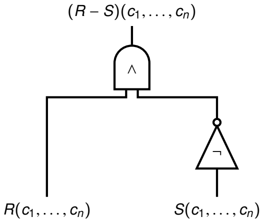
$\sigma_{i=j}(R)$ analogous.

$\pi_{a_1,\ldots,a_\ell}(R)$ for all tuples $\langle c_1, \ldots, c_n \rangle, \ldots, \langle c'_1, \ldots, c'_n \rangle$ in R with $c_{a_1} = c'_{a_1}, \ldots, c_{a_\ell} = c'_{a_\ell}$, we add the circuit:

$R \bowtie S$ for each tuple $\langle a_1, \ldots, a_\ell, c_1, \ldots, c_n \rangle$ in R and each tuple $\langle b_1, \ldots, b_k, c_1, \ldots, c_n \rangle$ in S, we add the circuit:

$\delta_{a_1,\ldots a_n \to b_1,\ldots,b_n}(R)$ for each tuple $\langle c_{a_1}, \ldots, c_{a_n} \rangle$ in R, we add the circuit:

$R - S$ for each tuple $\langle c_1, \ldots, c_n \rangle$ in R, we add the circuit:



Exercise 3

Exercise. Expressions of relational algebra under named perspective can be translated into Boolean circuits, in a similar fashion to the translation illustrated for FO queries in the lecture. Show how each operator of relational algebra gives rise to a corresponding circuit by describing the circuits for the following expressions:

$\sigma_{i=c}(R)$ (c a constant)	$\sigma_{i=j}(R)$ (j an attribute)
$\pi_{a_1,\dots,a_\ell}(R)$	$R \bowtie S$
$\delta_{a_1,\dots,a_\ell \to b_1,\dots,b_\ell}(R)$	$R - S$
$R \cup S$	$R \cap S$

Solution.

$\sigma_{i=c}(R)$ for each tuple $\langle c_1, \dots, c_n \rangle$ in R, we add one of these two circuits:

$\sigma_{i=j}(R)$ analogous.

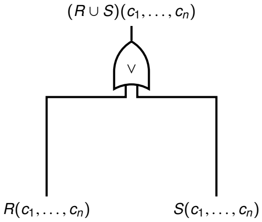
$\pi_{a_1,\dots,a_\ell}(R)$ for all tuples $\langle c_1, \dots, c_n \rangle, \dots, \langle c'_1, \dots, c'_n \rangle$ in R with $c_{a_1} = c'_{a_1}, \dots, c_{a_\ell} = c'_{a_\ell}$, we add the circuit:

$R \bowtie S$ for each tuple $\langle a_1, \dots, a_\ell, c_1, \dots, c_n \rangle$ in R and each tuple $\langle b_1, \dots, b_k, c_1, \dots, c_n \rangle$ in S, we add the circuit:

$\delta_{a_1,\dots a_n \to b_1,\dots,b_n}(R)$ for each tuple $\langle c_{a_1}, \dots, c_{a_n} \rangle$ in R, we add the circuit:

$R - S$ for each tuple $\langle c_1, \dots, c_n \rangle$ in R, we add the circuit:

$R \cup S$ for each tuple $\langle c_1, \dots, c_n \rangle$ in R, we add the circuit:



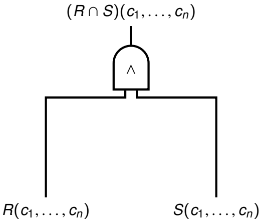
Exercise 3

Exercise. Expressions of relational algebra under named perspective can be translated into Boolean circuits, in a similar fashion to the translation illustrated for FO queries in the lecture. Show how each operator of relational algebra gives rise to a corresponding circuit by describing the circuits for the following expressions:

$\sigma_{i=c}(R)$ (c a constant)	$\sigma_{i=j}(R)$ (j an attribute)
$\pi_{a_1,\dots,a_\ell}(R)$	$R \bowtie S$
$\delta_{a_1,\dots,a_\ell \to b_1,\dots,b_\ell}(R)$	$R - S$
$R \cup S$	$R \cap S$

Solution.

- $\sigma_{i=c}(R)$ for each tuple $\langle c_1, \dots, c_n \rangle$ in R, we add one of these two circuits:
- $\sigma_{i=j}(R)$ analogous.
- $\pi_{a_1,\dots,a_\ell}(R)$ for all tuples $\langle c_1, \dots, c_n \rangle, \dots, \langle c'_1, \dots, c'_n \rangle$ in R with $c_{a_1} = c'_{a_1}, \dots, c_{a_\ell} = c'_{a_\ell}$, we add the circuit:
- $R \bowtie S$ for each tuple $\langle a_1, \dots, a_\ell, c_1, \dots, c_n \rangle$ in R and each tuple $\langle b_1, \dots, b_k, c_1, \dots, c_n \rangle$ in S, we add the circuit:
- $\delta_{a_1,\dots a_n \to b_1,\dots,b_n}(R)$ for each tuple $\langle c_{a_1}, \dots, c_{a_n} \rangle$ in R, we add the circuit:
- $R - S$ for each tuple $\langle c_1, \dots, c_n \rangle$ in R, we add the circuit:
- $R \cup S$ for each tuple $\langle c_1, \dots, c_n \rangle$ in R, we add the circuit:
- $R \cap S$ analogous to $R \bowtie S$.



Exercise 4

Exercise. Decide whether the following statements are true or false:

1. The combined complexity of a query language is at least as high as its data complexity.
2. The query complexity of a query language is at least as high as its data complexity.

If true, explain why, otherwise give a counter-example.

Exercise 4

Exercise. Decide whether the following statements are true or false:

1. The combined complexity of a query language is at least as high as its data complexity.
2. The query complexity of a query language is at least as high as its data complexity.

If true, explain why, otherwise give a counter-example.

Definition (Lecture 3, Slide 5)

Combined complexity given BCQ q and database instance $\mathcal{I}$ does $\mathcal{I} \models q$ hold?

Data complexity given database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold for a *fixed* BCQ q?

Query complexity given BCQ q, does $\mathcal{I} \models q$ hold for a *fixed* database instance $\mathcal{I}$?

Exercise 4

Exercise. Decide whether the following statements are true or false:

1. The combined complexity of a query language is at least as high as its data complexity.
2. The query complexity of a query language is at least as high as its data complexity.

If true, explain why, otherwise give a counter-example.

Definition (Lecture 3, Slide 5)

Combined complexity given BCQ q and database instance $\mathcal{I}$ does $\mathcal{I} \models q$ hold?

Data complexity given database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold for a *fixed* BCQ q?

Query complexity given BCQ q, does $\mathcal{I} \models q$ hold for a *fixed* database instance $\mathcal{I}$?

Solution.

Exercise 4

Exercise. Decide whether the following statements are true or false:

1. The combined complexity of a query language is at least as high as its data complexity.
2. The query complexity of a query language is at least as high as its data complexity.

If true, explain why, otherwise give a counter-example.

Definition (Lecture 3, Slide 5)

Combined complexity given BCQ q and database instance $\mathcal{I}$ does $\mathcal{I} \models q$ hold?

Data complexity given database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold for a *fixed* BCQ q?

Query complexity given BCQ q, does $\mathcal{I} \models q$ hold for a *fixed* database instance $\mathcal{I}$?

Solution.

1. True (*why?*).

Exercise 4

Exercise. Decide whether the following statements are true or false:

1. The combined complexity of a query language is at least as high as its data complexity.
2. The query complexity of a query language is at least as high as its data complexity.

If true, explain why, otherwise give a counter-example.

Definition (Lecture 3, Slide 5)

Combined complexity given BCQ q and database instance $\mathcal{I}$ does $\mathcal{I} \models q$ hold?

Data complexity given database instance $\mathcal{I}$, does $\mathcal{I} \models q$ hold for a *fixed* BCQ q?

Query complexity given BCQ q, does $\mathcal{I} \models q$ hold for a *fixed* database instance $\mathcal{I}$?

Solution.

1. True (*why?*).
2. False: Consider $L = \{q\}$ with q a non-trivial BCQ, i.e., a BCQ such that there are database instances $\mathcal{I}$ and $\mathcal{J}$ with $\mathcal{I} \models q$ and $\mathcal{J} \not\models q$. Then the query complexity is constant, yet the data complexity of L is still in AC^0.

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A $\textsc{LogSpace}$ transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a $\textsc{LogSpace}$ transducer is the contents of its output tape when it halts, i.e., $\textsc{LogSpace}$ transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- Let $f, g : \Sigma^* \to \Sigma^*$ be LogSpace-computable functions.

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- Let $f, g : \Sigma^* \to \Sigma^*$ be LogSpace-computable functions.
- Let $\mathcal{M}_f$ and $\mathcal{M}_g$ be LogSpace transducers computing f and g, respectively.

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A $\textsc{LogSpace}$ transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a $\textsc{LogSpace}$ transducer is the contents of its output tape when it halts, i.e., $\textsc{LogSpace}$ transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- Let $f, g : \Sigma^* \to \Sigma^*$ be $\textsc{LogSpace}$-computable functions.
- Let $\mathcal{M}_f$ and $\mathcal{M}_g$ be $\textsc{LogSpace}$ transducers computing f and g, respectively.
- We show that $f \circ g$ is also $\textsc{LogSpace}$ computable by constructing a $\textsc{LogSpace}$ transducer $\mathcal{M}$ computing $f \circ g$:

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A $\textsc{LogSpace}$ transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a $\textsc{LogSpace}$ transducer is the contents of its output tape when it halts, i.e., $\textsc{LogSpace}$ transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- Let $f, g : \Sigma^* \to \Sigma^*$ be $\textsc{LogSpace}$-computable functions.
- Let $\mathcal{M}_f$ and $\mathcal{M}_g$ be $\textsc{LogSpace}$ transducers computing f and g, respectively.
- We show that $f \circ g$ is also $\textsc{LogSpace}$ computable by constructing a $\textsc{LogSpace}$ transducer $\mathcal{M}$ computing $f \circ g$:
  1. We can't just simulate $\mathcal{M}_g$ to compute $g(w)$ for input w: $|g(w)|$ may be polynomial in $|w|$ (but not larger, since $\mathrm{L} \subseteq \mathrm{P}$).

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A $\textsc{LogSpace}$ transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a $\textsc{LogSpace}$ transducer is the contents of its output tape when it halts, i.e., $\textsc{LogSpace}$ transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- Let $f, g : \Sigma^* \to \Sigma^*$ be $\textsc{LogSpace}$-computable functions.
- Let $\mathcal{M}_f$ and $\mathcal{M}_g$ be $\textsc{LogSpace}$ transducers computing f and g, respectively.
- We show that $f \circ g$ is also $\textsc{LogSpace}$ computable by constructing a $\textsc{LogSpace}$ transducer $\mathcal{M}$ computing $f \circ g$:
  1. We can't just simulate $\mathcal{M}_g$ to compute $g(w)$ for input w: $|g(w)|$ may be polynomial in $|w|$ (but not larger, since $\mathrm{L} \subseteq \mathrm{P}$).
  2. But we can construct $\mathcal{M}'_g$ that computes the k-th symbol of $g(w)$:

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A LOGSPACE transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LOGSPACE transducer is the contents of its output tape when it halts, i.e., LOGSPACE transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- Let $f, g : \Sigma^* \to \Sigma^*$ be LOGSPACE-computable functions.
- Let $\mathcal{M}_f$ and $\mathcal{M}_g$ be LOGSPACE transducers computing f and g, respectively.
- We show that $f \circ g$ is also LOGSPACE computable by constructing a LOGSPACE transducer $\mathcal{M}$ computing $f \circ g$:
  1. We can't just simulate $\mathcal{M}_g$ to compute $g(w)$ for input w: $|g(w)|$ may be polynomial in $|w|$ (but not larger, since $\mathrm{L} \subseteq \mathrm{P}$).
  2. But we can construct $\mathcal{M}'_g$ that computes the k-th symbol of $g(w)$:
     - 2.1 We use a binary counter p to store k (since $|g(w)|$ is polynomial in $|w|$, we can do that in logarithmic space).

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- Let $f, g : \Sigma^* \to \Sigma^*$ be LogSpace-computable functions.
- Let $\mathcal{M}_f$ and $\mathcal{M}_g$ be LogSpace transducers computing f and g, respectively.
- We show that $f \circ g$ is also LogSpace computable by constructing a LogSpace transducer $\mathcal{M}$ computing $f \circ g$:
  1. We can't just simulate $\mathcal{M}_g$ to compute $g(w)$ for input w: $|g(w)|$ may be polynomial in $|w|$ (but not larger, since $\mathrm{L} \subseteq \mathrm{P}$).
  2. But we can construct $\mathcal{M}'_g$ that computes the k-th symbol of $g(w)$:
     - 2.1 We use a binary counter p to store k (since $|g(w)|$ is polynomial in $|w|$, we can do that in logarithmic space).
     - 2.2 On input $p\#w$, $\mathcal{M}'_g$ computes the k-th symbol of $g(w)$.

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A LOGSPACE transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LOGSPACE transducer is the contents of its output tape when it halts, i.e., LOGSPACE transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- Let $f, g : \Sigma^* \to \Sigma^*$ be LOGSPACE-computable functions.
- Let $\mathcal{M}_f$ and $\mathcal{M}_g$ be LOGSPACE transducers computing f and g, respectively.
- We show that $f \circ g$ is also LOGSPACE computable by constructing a LOGSPACE transducer $\mathcal{M}$ computing $f \circ g$:
  1. We can't just simulate $\mathcal{M}_g$ to compute $g(w)$ for input w: $|g(w)|$ may be polynomial in $|w|$ (but not larger, since $\mathrm{L} \subseteq \mathrm{P}$).
  2. But we can construct $\mathcal{M}'_g$ that computes the k-th symbol of $g(w)$:
     - 2.1 We use a binary counter p to store k (since $|g(w)|$ is polynomial in $|w|$, we can do that in logarithmic space).
     - 2.2 On input $p\#w$, $\mathcal{M}'_g$ computes the k-th symbol of $g(w)$.
  3. Then $\mathcal{M}$ computes $f \circ g$ on input w by simulating $\mathcal{M}_f$.

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A $\textsc{LogSpace}$ transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a $\textsc{LogSpace}$ transducer is the contents of its output tape when it halts, i.e., $\textsc{LogSpace}$ transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- Let $f, g : \Sigma^* \to \Sigma^*$ be $\textsc{LogSpace}$-computable functions.
- Let $\mathcal{M}_f$ and $\mathcal{M}_g$ be $\textsc{LogSpace}$ transducers computing f and g, respectively.
- We show that $f \circ g$ is also $\textsc{LogSpace}$ computable by constructing a $\textsc{LogSpace}$ transducer $\mathcal{M}$ computing $f \circ g$:
  1. We can't just simulate $\mathcal{M}_g$ to compute $g(w)$ for input w: $|g(w)|$ may be polynomial in $|w|$ (but not larger, since $\mathrm{L} \subseteq \mathrm{P}$).
  2. But we can construct $\mathcal{M}'_g$ that computes the k-th symbol of $g(w)$:
     - 2.1 We use a binary counter p to store k (since $|g(w)|$ is polynomial in $|w|$, we can do that in logarithmic space).
     - 2.2 On input $p\#w$, $\mathcal{M}'_g$ computes the k-th symbol of $g(w)$.
  3. Then $\mathcal{M}$ computes $f \circ g$ on input w by simulating $\mathcal{M}_f$.
  4. Each time the simulation of $\mathcal{M}_f$ tries to read the k-th symbol of $g(w)$, we simulate $\mathcal{M}'_g$, reading w from the input tape and p from the working tape, respectively, storing the result in a single cell of the working tape.

Exercise 5

Exercise. Show that the composition of logspace reductions yields a logspace reduction.

Definition (Lecture 3, Slides 20–21)

A LogSpace transducer is a deterministic TM with three tapes:

- a read-only input tape
- a read/write working tape of size $O(\log n)$
- a write-only, write-once output tape

The output of a LogSpace transducer is the contents of its output tape when it halts, i.e., LogSpace transducers compute partial functions $\Sigma^* \to \Sigma^*$.

Solution.

- Let $f, g : \Sigma^* \to \Sigma^*$ be LogSpace-computable functions.
- Let $\mathcal{M}_f$ and $\mathcal{M}_g$ be LogSpace transducers computing f and g, respectively.
- We show that $f \circ g$ is also LogSpace computable by constructing a LogSpace transducer $\mathcal{M}$ computing $f \circ g$:
  1. We can't just simulate $\mathcal{M}_g$ to compute $g(w)$ for input w: $|g(w)|$ may be polynomial in $|w|$ (but not larger, since $\mathrm{L} \subseteq \mathrm{P}$).
  2. But we can construct $\mathcal{M}'_g$ that computes the k-th symbol of $g(w)$:
     - 2.1 We use a binary counter p to store k (since $|g(w)|$ is polynomial in $|w|$, we can do that in logarithmic space).
     - 2.2 On input $p\#w$, $\mathcal{M}'_g$ computes the k-th symbol of $g(w)$.
  3. Then $\mathcal{M}$ computes $f \circ g$ on input w by simulating $\mathcal{M}_f$.
  4. Each time the simulation of $\mathcal{M}_f$ tries to read the k-th symbol of $g(w)$, we simulate $\mathcal{M}'_g$, reading w from the input tape and p from the working tape, respectively, storing the result in a single cell of the working tape.
  5. Both simulations can be performed in logarithmic space, and thus, $\mathcal{M}$ runs in logarithmic space.

Exercise 6

Exercise. Is the question "$\mathrm{P} = \mathrm{NP}$?" decidable?

Exercise 6

Exercise. Is the question "$\mathrm{P} = \mathrm{NP}$?" decidable?

Definition (Lecture 3, slide 10)

A TM decides a decision problem $\mathcal{L}$ if it halts on all inputs and accepts exactly the words in $\mathcal{L}$.

Exercise 6

Exercise. Is the question “$\mathrm{P} = \mathrm{NP}$?” decidable?

Definition (Lecture 3, slide 10)

A TM decides a decision problem $\mathcal{L}$ if it halts on all inputs and accepts exactly the words in $\mathcal{L}$.

Solution.

Exercise 6

Exercise. Is the question "$\mathrm{P} = \mathrm{NP}$?" decidable?

Definition (Lecture 3, slide 10)

A TM decides a decision problem $\mathcal{L}$ if it halts on all inputs and accepts exactly the words in $\mathcal{L}$.

Solution.

- Let $\mathcal{L}$ be the decision problem for "$\mathrm{P} = \mathrm{NP}$?", i.e., let $\mathcal{L} = \Sigma^*$ if $\mathrm{P} = \mathrm{NP}$, and let $\mathcal{L} = \emptyset$ otherwise.

Exercise 6

Exercise. Is the question "$\mathrm{P} = \mathrm{NP}$?" decidable?

Definition (Lecture 3, slide 10)

A TM decides a decision problem $\mathcal{L}$ if it halts on all inputs and accepts exactly the words in $\mathcal{L}$.

Solution.

- Let $\mathcal{L}$ be the decision problem for "$\mathrm{P} = \mathrm{NP}$?", i.e., let $\mathcal{L} = \Sigma^*$ if $\mathrm{P} = \mathrm{NP}$, and let $\mathcal{L} = \emptyset$ otherwise.
- Let $\mathcal{M}_A$ and $\mathcal{M}_R$ be two terminating TMs that accept and reject every input, respectively.

Exercise 6

Exercise. Is the question "$\mathrm{P} = \mathrm{NP}$?" decidable?

Definition (Lecture 3, slide 10)

A TM decides a decision problem $\mathcal{L}$ if it halts on all inputs and accepts exactly the words in $\mathcal{L}$.

Solution.

- Let $\mathcal{L}$ be the decision problem for "$\mathrm{P} = \mathrm{NP}$?", i.e., let $\mathcal{L} = \Sigma^*$ if $\mathrm{P} = \mathrm{NP}$, and let $\mathcal{L} = \emptyset$ otherwise.
- Let $\mathcal{M}_A$ and $\mathcal{M}_R$ be two terminating TMs that accept and reject every input, respectively.
- One of these two TMs decides $\mathcal{L}$.

Exercise 6

Exercise. Is the question "$\mathrm{P} = \mathrm{NP}$?" decidable?

Definition (Lecture 3, slide 10)

A TM decides a decision problem $\mathcal{L}$ if it halts on all inputs and accepts exactly the words in $\mathcal{L}$.

Solution.

- Let $\mathcal{L}$ be the decision problem for "$\mathrm{P} = \mathrm{NP}$?", i.e., let $\mathcal{L} = \Sigma^*$ if $\mathrm{P} = \mathrm{NP}$, and let $\mathcal{L} = \emptyset$ otherwise.
- Let $\mathcal{M}_A$ and $\mathcal{M}_R$ be two terminating TMs that accept and reject every input, respectively.
- One of these two TMs decides $\mathcal{L}$.
- Thus, $\mathcal{L}$ is decidable, and hence, so is "$\mathrm{P} = \mathrm{NP}$?".