Exercise 6: Trakhtenbrot's Theorem

Database Theory

2025-05-20

Lukas Gerlach, Maximilian Marx, Markus Krötzsch

Exercise 1

Exercise. Use Trakhtenbrot's Theorem to show that the following problems are undecidable by reducing finite satisfiability to each of them:

1. FO query containment.
2. FO query emptiness.
3. Domain independence of FO queries.

Exercise 1

Exercise. Use Trakhtenbrot's Theorem to show that the following problems are undecidable by reducing finite satisfiability to each of them:

1. FO query containment.
2. FO query emptiness.
3. Domain independence of FO queries.

Theorem (Trakhtenbrot's Theorem, Lecture 9, Slide 9)

Finite-model reasoning of first-order logic is undecidable.

Exercise 1

Exercise. Use Trakhtenbrot's Theorem to show that the following problems are undecidable by reducing finite satisfiability to each of them:

1. FO query containment.
2. FO query emptiness.
3. Domain independence of FO queries.

Theorem (Trakhtenbrot's Theorem, Lecture 9, Slide 9)

Finite-model reasoning of first-order logic is undecidable.

Solution.

Exercise 1

Exercise. Use Trakhtenbrot's Theorem to show that the following problems are undecidable by reducing finite satisfiability to each of them:

1. FO query containment.
2. FO query emptiness.
3. Domain independence of FO queries.

Theorem (Trakhtenbrot's Theorem, Lecture 9, Slide 9)

Finite-model reasoning of first-order logic is undecidable.

Solution.

1. - Let ψ be some unsatisfiable Boolean query, e.g., let $\psi = \exists x.\, A(x) \wedge \neg A(x)$.

Exercise 1

Exercise. Use Trakhtenbrot's Theorem to show that the following problems are undecidable by reducing finite satisfiability to each of them:

1. FO query containment.
2. FO query emptiness.
3. Domain independence of FO queries.

Theorem (Trakhtenbrot's Theorem, Lecture 9, Slide 9)

Finite-model reasoning of first-order logic is undecidable.

Solution.

1.
   - Let ψ be some unsatisfiable Boolean query, e.g., let $\psi = \exists x.\, A(x) \wedge \neg A(x)$.
   - A BQ φ is finitely satisfiable iff $\varphi \not\sqsubseteq \psi$.

Exercise 1

Exercise. Use Trakhtenbrot's Theorem to show that the following problems are undecidable by reducing finite satisfiability to each of them:

1. FO query containment.
2. FO query emptiness.
3. Domain independence of FO queries.

Theorem (Trakhtenbrot's Theorem, Lecture 9, Slide 9)

Finite-model reasoning of first-order logic is undecidable.

Solution.

1. - Let ψ be some unsatisfiable Boolean query, e.g., let $\psi = \exists x.\, A(x) \wedge \neg A(x)$.
   - A BQ φ is finitely satisfiable iff $\varphi \not\sqsubseteq \psi$.
2. - A query φ is empty iff $M[\varphi](\mathcal{I}) = \emptyset$ for every database instance $\mathcal{I}$.

Exercise 1

Exercise. Use Trakhtenbrot's Theorem to show that the following problems are undecidable by reducing finite satisfiability to each of them:

1. FO query containment.
2. FO query emptiness.
3. Domain independence of FO queries.

Theorem (Trakhtenbrot's Theorem, Lecture 9, Slide 9)

Finite-model reasoning of first-order logic is undecidable.

Solution.

1. - Let ψ be some unsatisfiable Boolean query, e.g., let $\psi = \exists x.\, A(x) \wedge \neg A(x)$.
   - A BQ φ is finitely satisfiable iff $\varphi \not\sqsubseteq \psi$.
2. - A query φ is empty iff $M[\varphi](\mathcal{I}) = \emptyset$ for every database instance $\mathcal{I}$.
   - A query φ is empty iff it is finitely unsatisfiable.

Exercise 1

Exercise. Use Trakhtenbrot's Theorem to show that the following problems are undecidable by reducing finite satisfiability to each of them:

1. FO query containment.
2. FO query emptiness.
3. Domain independence of FO queries.

Theorem (Trakhtenbrot's Theorem, Lecture 9, Slide 9)

Finite-model reasoning of first-order logic is undecidable.

Solution.

1. - Let ψ be some unsatisfiable Boolean query, e.g., let $\psi = \exists x.\, A(x) \land \neg A(x)$.
   - A BQ φ is finitely satisfiable iff $\varphi \not\sqsubseteq \psi$.
2. - A query φ is empty iff $M[\varphi](\mathcal{I}) = \emptyset$ for every database instance $\mathcal{I}$.
   - A query φ is empty iff it is finitely unsatisfiable.
3. - A query $\varphi[\mathbf{x}]$ is domain independent iff the answers over a database instance $\mathcal{I}$ are independent of the domain $\Delta^{\mathcal{I}}$.

Exercise 1

Exercise. Use Trakhtenbrot's Theorem to show that the following problems are undecidable by reducing finite satisfiability to each of them:

1. FO query containment.
2. FO query emptiness.
3. Domain independence of FO queries.

Theorem (Trakhtenbrot's Theorem, Lecture 9, Slide 9)

Finite-model reasoning of first-order logic is undecidable.

Solution.

1.
   - Let ψ be some unsatisfiable Boolean query, e.g., let $\psi = \exists x.\, A(x) \land \neg A(x)$.
   - A BQ φ is finitely satisfiable iff $\varphi \not\sqsubseteq \psi$.
2.
   - A query φ is empty iff $M[\varphi](\mathcal{I}) = \emptyset$ for every database instance $\mathcal{I}$.
   - A query φ is empty iff it is finitely unsatisfiable.
3.
   - A query $\varphi[\mathbf{x}]$ is domain independent iff the answers over a database instance $\mathcal{I}$ are independent of the domain $\Delta^{\mathcal{I}}$.
   - A query $\varphi[\mathbf{x}]$ is empty iff $\neg R(y) \land \forall \mathbf{x}.\, \varphi$ is domain independent, where R is a fresh unary relation and y is a fresh variable.

Exercise 2

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
For each of the following statements, decide if it is true or false. Justify your answer in each case by explaining why the statement does (or does not) follow from the formula.

1. If the formula has a model at all, then this model is finite.
2. Every model contains a “start configuration”: a right-sequence of elements (“cells”) that are not reachable from any other cell via future, and where there is a first element in the chain (i.e., a cell with no element to its left).
3. Every model contains exactly one such start configuration.
4. If a cell is reachable from the first cell of the start configuration via future, then it does not have a cell on its left.
5. The future of a cell’s neighbour is equal to the neighbour of the cell’s future.
6. If the Turing machine halts on the input, then every model of the formula is finite.
7. No cell can ever reach itself via future, i.e., there is no loop in the future relation.

Exercise 2

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
For each of the following statements, decide if it is true or false. Justify your answer in each case by explaining why the statement does (or does not) follow from the formula.

1. If the formula has a model at all, then this model is finite.
2. Every model contains a “start configuration”: a right-sequence of elements (“cells”) that are not reachable from any other cell via future, and where there is a first element in the chain (i.e., a cell with no element to its left).
3. Every model contains exactly one such start configuration.
4. If a cell is reachable from the first cell of the start configuration via future, then it does not have a cell on its left.
5. The future of a cell’s neighbour is equal to the neighbour of the cell’s future.
6. If the Turing machine halts on the input, then every model of the formula is finite.
7. No cell can ever reach itself via future, i.e., there is no loop in the future relation.

Solution.

Exercise 2

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
For each of the following statements, decide if it is true or false. Justify your answer in each case by explaining why the statement does (or does not) follow from the formula.

1. If the formula has a model at all, then this model is finite.
2. Every model contains a “start configuration”: a right-sequence of elements (“cells”) that are not reachable from any other cell via future, and where there is a first element in the chain (i.e., a cell with no element to its left).
3. Every model contains exactly one such start configuration.
4. If a cell is reachable from the first cell of the start configuration via future, then it does not have a cell on its left.
5. The future of a cell’s neighbour is equal to the neighbour of the cell’s future.
6. If the Turing machine halts on the input, then every model of the formula is finite.
7. No cell can ever reach itself via future, i.e., there is no loop in the future relation.

Solution.

1. False. If the TM does not halt, the formula has an infinite model, but no finite models.

Exercise 2

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
For each of the following statements, decide if it is true or false. Justify your answer in each case by explaining why the statement does (or does not) follow from the formula.

1. If the formula has a model at all, then this model is finite.
2. Every model contains a "start configuration": a right-sequence of elements ("cells") that are not reachable from any other cell via future, and where there is a first element in the chain (i.e., a cell with no element to its left).
3. Every model contains exactly one such start configuration.
4. If a cell is reachable from the first cell of the start configuration via future, then it does not have a cell on its left.
5. The future of a cell's neighbour is equal to the neighbour of the cell's future.
6. If the Turing machine halts on the input, then every model of the formula is finite.
7. No cell can ever reach itself via future, i.e., there is no loop in the future relation.

Solution.

1. False. If the TM does not halt, the formula has an infinite model, but no finite models.
2. True.

$$\varphi_w = \exists x_1, \dots, x_n.\ H_{q_{\text{start}}}(x_1) \wedge \neg\exists z.\ \text{right}(z, x_1) \wedge S_{\sigma_1}(x_1) \wedge \neg\exists z.\ \text{future}(z, x_1) \wedge \text{right}(x_1, x_2) \wedge \cdots \wedge S_{\sigma_n}(x_n) \wedge \neg\exists z.\ \text{future}(z, x_n) \wedge \forall y.\ \big(\text{right}^+(x_n, y) \rightarrow (S_{\sqcup}(y) \wedge \neg\exists z.\ \text{future}(z, y))\big)$$

Exercise 2

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
For each of the following statements, decide if it is true or false. Justify your answer in each case by explaining why the statement does (or does not) follow from the formula.

1. If the formula has a model at all, then this model is finite.
2. Every model contains a “start configuration”: a right-sequence of elements (“cells”) that are not reachable from any other cell via future, and where there is a first element in the chain (i.e., a cell with no element to its left).
3. Every model contains exactly one such start configuration.
4. If a cell is reachable from the first cell of the start configuration via future, then it does not have a cell on its left.
5. The future of a cell’s neighbour is equal to the neighbour of the cell’s future.
6. If the Turing machine halts on the input, then every model of the formula is finite.
7. No cell can ever reach itself via future, i.e., there is no loop in the future relation.

Solution.

3. False. Take two isomorphic copies of a model side-by-side.

Exercise 2

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
For each of the following statements, decide if it is true or false. Justify your answer in each case by explaining why the statement does (or does not) follow from the formula.

1. If the formula has a model at all, then this model is finite.
2. Every model contains a "start configuration": a right-sequence of elements ("cells") that are not reachable from any other cell via future, and where there is a first element in the chain (i.e., a cell with no element to its left).
3. Every model contains exactly one such start configuration.
4. If a cell is reachable from the first cell of the start configuration via future, then it does not have a cell on its left.
5. The future of a cell's neighbour is equal to the neighbour of the cell's future.
6. If the Turing machine halts on the input, then every model of the formula is finite.
7. No cell can ever reach itself via future, i.e., there is no loop in the future relation.

Solution.

3. False. Take two isomorphic copies of a model side-by-side.
4. True.

$$\begin{aligned}
\varphi_{fp1} &= \forall x_2, y_1.\ (\exists x_1.\ \mathrm{right}(x_1, y_1) \wedge \mathrm{future}(x_1, x_2)) \leftrightarrow (\exists y_2.\ \mathrm{future}(y_1, y_2) \wedge \mathrm{right}(x_2, y_2)) \\
\varphi_{fp2} &= \forall x_1, y_2.\ (\exists y_1.\ \mathrm{right}(x_1, y_1) \wedge \mathrm{future}(y_1, y_2)) \leftrightarrow (\exists x_2.\ \mathrm{future}(x_1, x_2) \wedge \mathrm{right}(x_2, y_2)) \\
\varphi_w &= \exists x_1, \ldots, x_n.\ H_{q_{\mathrm{start}}}(x_1) \wedge \neg\exists z.\ \mathrm{right}(z, x_1) \wedge S_{\sigma_1}(x_1) \wedge \neg\exists z.\ \mathrm{future}(z, x_1) \wedge \mathrm{right}(x_1, x_2) \wedge \cdots
\end{aligned}$$

Exercise 2

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
For each of the following statements, decide if it is true or false. Justify your answer in each case by explaining why the statement does (or does not) follow from the formula.

1. If the formula has a model at all, then this model is finite.
2. Every model contains a "start configuration": a right-sequence of elements ("cells") that are not reachable from any other cell via future, and where there is a first element in the chain (i.e., a cell with no element to its left).
3. Every model contains exactly one such start configuration.
4. If a cell is reachable from the first cell of the start configuration via future, then it does not have a cell on its left.
5. The future of a cell's neighbour is equal to the neighbour of the cell's future.
6. If the Turing machine halts on the input, then every model of the formula is finite.
7. No cell can ever reach itself via future, i.e., there is no loop in the future relation.

Solution.

5. True.

$$\varphi_r = \forall x, y, y'.\ \mathrm{right}(x, y) \land \mathrm{right}(x, y') \rightarrow y \approx y' \qquad \varphi_l = \forall x, x', y.\ \mathrm{right}(x, y) \land \mathrm{right}(x', y) \rightarrow x \approx x'$$

$$\varphi_f = \forall x, y, y'.\ \mathrm{future}(x, y) \land \mathrm{future}(x, y') \rightarrow y \approx y' \qquad \varphi_p = \forall x, x', y.\ \mathrm{future}(x, y) \land \mathrm{future}(x', y) \rightarrow x \approx x'$$

Exercise 2

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
For each of the following statements, decide if it is true or false. Justify your answer in each case by explaining why the statement does (or does not) follow from the formula.

1. If the formula has a model at all, then this model is finite.
2. Every model contains a "start configuration": a right-sequence of elements ("cells") that are not reachable from any other cell via future, and where there is a first element in the chain (i.e., a cell with no element to its left).
3. Every model contains exactly one such start configuration.
4. If a cell is reachable from the first cell of the start configuration via future, then it does not have a cell on its left.
5. The future of a cell's neighbour is equal to the neighbour of the cell's future.
6. If the Turing machine halts on the input, then every model of the formula is finite.
7. No cell can ever reach itself via future, i.e., there is no loop in the future relation.

Solution.

5. True.

$$\varphi_r = \forall x, y, y'.\, \text{right}(x, y) \land \text{right}(x, y') \to y \approx y' \qquad \varphi_l = \forall x, x', y.\, \text{right}(x, y) \land \text{right}(x', y) \to x \approx x'$$

$$\varphi_f = \forall x, y, y'.\, \text{future}(x, y) \land \text{future}(x, y') \to y \approx y' \qquad \varphi_p = \forall x, x', y.\, \text{future}(x, y) \land \text{future}(x', y) \to x \approx x'$$

6. False. Recall that, by the Compactness theorem, any FO formula that has arbitrarily large finite models also has an infinite model.

Exercise 2

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
For each of the following statements, decide if it is true or false. Justify your answer in each case by explaining why the statement does (or does not) follow from the formula.

1. If the formula has a model at all, then this model is finite.
2. Every model contains a "start configuration": a right-sequence of elements ("cells") that are not reachable from any other cell via future, and where there is a first element in the chain (i.e., a cell with no element to its left).
3. Every model contains exactly one such start configuration.
4. If a cell is reachable from the first cell of the start configuration via future, then it does not have a cell on its left.
5. The future of a cell's neighbour is equal to the neighbour of the cell's future.
6. If the Turing machine halts on the input, then every model of the formula is finite.
7. No cell can ever reach itself via future, i.e., there is no loop in the future relation.

Solution.

5. True.

$$\varphi_r = \forall x, y, y'.\ \mathrm{right}(x, y) \wedge \mathrm{right}(x, y') \to y \approx y' \qquad \varphi_l = \forall x, x', y.\ \mathrm{right}(x, y) \wedge \mathrm{right}(x', y) \to x \approx x'$$

$$\varphi_f = \forall x, y, y'.\ \mathrm{future}(x, y) \wedge \mathrm{future}(x, y') \to y \approx y' \qquad \varphi_p = \forall x, x', y.\ \mathrm{future}(x, y) \wedge \mathrm{future}(x', y) \to x \approx x'$$

6. False. Recall that, by the Compactness theorem, any FO formula that has arbitrarily large finite models also has an infinite model.
7. False. Take a model, and add a fact $\mathrm{future}(\star, \star)$ with $\star$ a fresh domain element.

Exercise 3

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
Extend this definition so that the resulting formula is finitely satisfiable if and only if:

1. a given non-deterministic TM halts after finitely many steps on a given input.
2. a given DTM halts after at most n steps (for a given number n).
3. a given DTM halts after at most 2^n steps (for a given number n).

Make sure that your encoding is polynomial in n.

Exercise 3

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
Extend this definition so that the resulting formula is finitely satisfiable if and only if:

1. a given non-deterministic TM halts after finitely many steps on a given input.
2. a given DTM halts after at most n steps (for a given number n).
3. a given DTM halts after at most 2^n steps (for a given number n).

Make sure that your encoding is polynomial in n.
Solution.

Exercise 3

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
Extend this definition so that the resulting formula is finitely satisfiable if and only if:

1. a given non-deterministic TM halts after finitely many steps on a given input.
2. a given DTM halts after at most n steps (for a given number n).
3. a given DTM halts after at most 2^n steps (for a given number n).

Make sure that your encoding is polynomial in n.
Solution.

1. - First, we normalise the NTM so that every non-deterministic transition defined by Δ is non-moving.

Exercise 3

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
Extend this definition so that the resulting formula is finitely satisfiable if and only if:

1. a given non-deterministic TM halts after finitely many steps on a given input.
2. a given DTM halts after at most *n* steps (for a given number *n*).
3. a given DTM halts after at most 2^n steps (for a given number *n*).

Make sure that your encoding is polynomial in *n*.
Solution.

1. - First, we normalise the NTM so that every non-deterministic transition defined by Δ is non-moving.
   - For every non-deterministic transition $\{\langle q, \sigma, q_1, \sigma_1, s\rangle, \ldots, \langle q, \sigma, q_n, \sigma_n, s\rangle\} \subseteq \Delta$, we add the following rule:
     $\varphi_\delta = \forall x.\, H_q(x) \wedge S_\sigma(x) \rightarrow \exists y.\, \text{future}(x, y) \wedge \left(\bigvee_{1 \leq i \leq n} \left(H_{q_i}(y) \wedge S_{\sigma_i}(y)\right)\right)$

Exercise 3

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
Extend this definition so that the resulting formula is finitely satisfiable if and only if:

1. a given non-deterministic TM halts after finitely many steps on a given input.
2. a given DTM halts after at most n steps (for a given number n).
3. a given DTM halts after at most 2^n steps (for a given number n).

Make sure that your encoding is polynomial in n.
Solution.

1.
   - First, we normalise the NTM so that every non-deterministic transition defined by Δ is non-moving.
   - For every non-deterministic transition $\{\langle q, \sigma, q_1, \sigma_1, s\rangle, \dots, \langle q, \sigma, q_n, \sigma_n, s\rangle\} \subseteq \Delta$, we add the following rule: $\varphi_\delta = \forall x.\, H_q(x) \wedge S_\sigma(x) \to \exists y.\, \mathrm{future}(x, y) \wedge \left(\bigvee_{1 \leq i \leq n} \left(H_{q_i}(y) \wedge S_{\sigma_i}(y)\right)\right)$
2.
   - Modify start configuration

$$\varphi_w = \exists \mathbf{x}.\, H_{q_{\mathrm{start}}}(x_1) \wedge C_1(x_1) \wedge \neg\exists z.\, \mathrm{right}(z, x_1) \wedge S_{\sigma_i}(x_i) \wedge \neg\exists z.\, \mathrm{future}(z, x_i)$$
$$\wedge\, \mathrm{right}(x_i, x_{i+1}) \wedge \forall y.\, \left(\mathrm{right}^+(x_n, y) \to (S\_(y) \wedge \neg\exists z.\, \mathrm{future}(z, y))\right)$$

Exercise 3

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
Extend this definition so that the resulting formula is finitely satisfiable if and only if:

1. a given non-deterministic TM halts after finitely many steps on a given input.
2. a given DTM halts after at most n steps (for a given number n).
3. a given DTM halts after at most 2^n steps (for a given number n).

Make sure that your encoding is polynomial in n.
Solution.

1.
   - First, we normalise the NTM so that every non-deterministic transition defined by Δ is non-moving.
   - For every non-deterministic transition $\{\langle q, \sigma, q_1, \sigma_1, s\rangle, \dots, \langle q, \sigma, q_n, \sigma_n, s\rangle\} \subseteq \Delta$, we add the following rule: $\varphi_\delta = \forall x.\ H_q(x) \wedge S_\sigma(x) \rightarrow \exists y.\ \mathrm{future}(x, y) \wedge \left(\bigvee_{1 \le i \le n} \left(H_{q_i}(y) \wedge S_{\sigma_i}(y)\right)\right)$
2.
   - Modify start configuration

$$\varphi_w = \exists \mathbf{x}.\ H_{q_{\text{start}}}(x_1) \wedge C_1(x_1) \wedge \neg\exists z.\ \mathrm{right}(z, x_1) \wedge S_{\sigma_i}(x_i) \wedge \neg\exists z.\ \mathrm{future}(z, x_i)$$
$$\wedge\ \mathrm{right}(x_i, x_{i+1}) \wedge \forall y.\ \left(\mathrm{right}^+(x_n, y) \rightarrow (S\_(y) \wedge \neg\exists z.\ \mathrm{future}(z, y))\right)$$

   - For all $i \in \{1, \dots, n\}$, add $\forall x, y.\ C_i(x) \wedge \mathrm{future}(x, y) \rightarrow C_{i+1}(y)$

Exercise 3

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
Extend this definition so that the resulting formula is finitely satisfiable if and only if:

1. a given non-deterministic TM halts after finitely many steps on a given input.
2. a given DTM halts after at most n steps (for a given number n).
3. a given DTM halts after at most 2^n steps (for a given number n).

Make sure that your encoding is polynomial in n.
Solution.

1.
   - First, we normalise the NTM so that every non-deterministic transition defined by Δ is non-moving.
   - For every non-deterministic transition $\{\langle q, \sigma, q_1, \sigma_1, s\rangle, \ldots, \langle q, \sigma, q_n, \sigma_n, s\rangle\} \subseteq \Delta$, we add the following rule:
     $\varphi_\delta = \forall x.\ H_q(x) \wedge S_\sigma(x) \rightarrow \exists y.\ \mathrm{future}(x, y) \wedge \left(\bigvee_{1 \leq i \leq n} \left(H_{q_i}(y) \wedge S_{\sigma_i}(y)\right)\right)$
2.
   - Modify start configuration

$$\varphi_w = \exists \mathbf{x}.\ H_{q_{\mathrm{start}}}(x_1) \wedge C_1(x_1) \wedge \neg\exists z.\ \mathrm{right}(z, x_1) \wedge S_{\sigma_i}(x_i) \wedge \neg\exists z.\ \mathrm{future}(z, x_i)$$
$$\wedge\ \mathrm{right}(x_i, x_{i+1}) \wedge \forall y.\ \left(\mathrm{right}^+(x_n, y) \rightarrow (S\_(y) \wedge \neg\exists z.\ \mathrm{future}(z, y))\right)$$

   - For all $i \in \{1, \ldots, n\}$, add $\forall x, y.\ C_i(x) \wedge \mathrm{future}(x, y) \rightarrow C_{i+1}(y)$
   - Add $\forall x.\ \neg C_{n+1}(x)$

Exercise 3

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
Extend this definition so that the resulting formula is finitely satisfiable if and only if:

1. a given non-deterministic TM halts after finitely many steps on a given input.
2. a given DTM halts after at most n steps (for a given number n).
3. a given DTM halts after at most 2^n steps (for a given number n).

Make sure that your encoding is polynomial in n.
Solution.

3. - Modify start configuration

$$\varphi_w = \exists \mathbf{x}.\ H_{q_{\text{start}}}(x_1) \wedge \neg B_1(x_1) \wedge \cdots \wedge \neg B_n(x_1) \wedge \neg\exists z.\ \text{right}(z, x_1) \wedge S_{\sigma_i}(x_i) \wedge \neg\exists z.\ \text{future}(z, x_i)$$
$$\wedge\ \text{right}(x_i, x_{i+1}) \wedge \forall y.\ \big(\text{right}^+(x_n, y) \rightarrow (S\_(y) \wedge \neg\exists z.\ \text{future}(z, y))\big)$$

Exercise 3

Exercise. In the lecture, we have seen a logical formula that is finitely satisfiable if and only if the given deterministic Turing machine (DTM) halts after finitely many steps on the given input.
Extend this definition so that the resulting formula is finitely satisfiable if and only if:

1. a given non-deterministic TM halts after finitely many steps on a given input.
2. a given DTM halts after at most n steps (for a given number n).
3. a given DTM halts after at most 2^n steps (for a given number n).

Make sure that your encoding is polynomial in n.
Solution.

3. - Modify start configuration

$$\varphi_w = \exists \mathbf{x}.\ H_{q_{\text{start}}}(x_1) \wedge \neg B_1(x_1) \wedge \cdots \wedge \neg B_n(x_1) \wedge \neg\exists z.\ \text{right}(z, x_1) \wedge S_{\sigma_i}(x_i) \wedge \neg\exists z.\ \text{future}(z, x_i)$$
$$\wedge\ \text{right}(x_i, x_{i+1}) \wedge \forall y.\ \big(\text{right}^+(x_n, y) \to (S\_(y) \wedge \neg\exists z.\ \text{future}(z, y))\big)$$

   - Add the following rules:

$$\neg B_n(x) \wedge \text{future}(x,y) \to B_n(y)$$
$$\neg B_{n-1}(x) \wedge B_n(x) \wedge \text{future}(x,y) \to B_{n-1}(y) \wedge \neg B_n(y)$$
$$\neg B_{n-2}(x) \wedge B_{n-1}(x) \wedge B_n(x) \wedge \text{future}(x,y) \to B_{n-2}(y) \wedge \neg B_{n-1}(y) \wedge \neg B_n(y)$$
$$\vdots$$
$$\neg(\exists x. B_1(x) \wedge \ldots \wedge B_n(x))$$

Exercise 4

Exercise. Apply the CQ minimisation algorithm to find a core of the following CQs:

1. $\exists x, y, z.\, \mathsf{R}(x, y) \land \mathsf{R}(x, z)$.
2. $\exists x, y, z.\, \mathsf{R}(x, y) \land \mathsf{R}(x, z) \land \mathsf{R}(y, z)$.
3. $\exists x, y, z.\, \mathsf{R}(x, y) \land \mathsf{R}(x, z) \land \mathsf{R}(y, z) \land \mathsf{R}(x, x)$.
4. $\exists v, w.\, \mathsf{S}(x, a, y) \land \mathsf{S}(x, v, y) \land \mathsf{S}(x, w, y) \land \mathsf{S}(x, x, x)$.

Exercise 4

Exercise. Apply the CQ minimisation algorithm to find a core of the following CQs:

1. $\exists x, y, z.\ R(x, y) \wedge R(x, z)$.
2. $\exists x, y, z.\ R(x, y) \wedge R(x, z) \wedge R(y, z)$.
3. $\exists x, y, z.\ R(x, y) \wedge R(x, z) \wedge R(y, z) \wedge R(x, x)$.
4. $\exists v, w.\ S(x, a, y) \wedge S(x, v, y) \wedge S(x, w, y) \wedge S(x, x, x)$.

Solution.

Exercise 4

Exercise. Apply the CQ minimisation algorithm to find a core of the following CQs:

1. $\exists x, y, z.\ \mathrm{R}(x, y) \wedge \mathrm{R}(x, z)$.
2. $\exists x, y, z.\ \mathrm{R}(x, y) \wedge \mathrm{R}(x, z) \wedge \mathrm{R}(y, z)$.
3. $\exists x, y, z.\ \mathrm{R}(x, y) \wedge \mathrm{R}(x, z) \wedge \mathrm{R}(y, z) \wedge \mathrm{R}(x, x)$.
4. $\exists v, w.\ \mathrm{S}(x, a, y) \wedge \mathrm{S}(x, v, y) \wedge \mathrm{S}(x, w, y) \wedge \mathrm{S}(x, x, x)$.

Solution.

1. $\exists x, y.\ \mathrm{R}(x, y)$.

Exercise 4

Exercise. Apply the CQ minimisation algorithm to find a core of the following CQs:

1. $\exists x, y, z.\ R(x, y) \wedge R(x, z)$.
2. $\exists x, y, z.\ R(x, y) \wedge R(x, z) \wedge R(y, z)$.
3. $\exists x, y, z.\ R(x, y) \wedge R(x, z) \wedge R(y, z) \wedge R(x, x)$.
4. $\exists v, w.\ S(x, a, y) \wedge S(x, v, y) \wedge S(x, w, y) \wedge S(x, x, x)$.

Solution.

1. $\exists x, y.\ R(x, y)$.
2. $\exists x, y, z.\ R(x, y) \wedge R(x, z) \wedge R(y, z)$.

Exercise 4

Exercise. Apply the CQ minimisation algorithm to find a core of the following CQs:

1. $\exists x, y, z.\ \mathrm{R}(x, y) \wedge \mathrm{R}(x, z)$.
2. $\exists x, y, z.\ \mathrm{R}(x, y) \wedge \mathrm{R}(x, z) \wedge \mathrm{R}(y, z)$.
3. $\exists x, y, z.\ \mathrm{R}(x, y) \wedge \mathrm{R}(x, z) \wedge \mathrm{R}(y, z) \wedge \mathrm{R}(x, x)$.
4. $\exists v, w.\ \mathrm{S}(x, a, y) \wedge \mathrm{S}(x, v, y) \wedge \mathrm{S}(x, w, y) \wedge \mathrm{S}(x, x, x)$.

Solution.

1. $\exists x, y.\ \mathrm{R}(x, y)$.
2. $\exists x, y, z.\ \mathrm{R}(x, y) \wedge \mathrm{R}(x, z) \wedge \mathrm{R}(y, z)$.
3. $\exists x.\ \mathrm{R}(x, x)$.

Exercise 4

Exercise. Apply the CQ minimisation algorithm to find a core of the following CQs:

1. $\exists x, y, z.\ \mathsf{R}(x, y) \land \mathsf{R}(x, z)$.
2. $\exists x, y, z.\ \mathsf{R}(x, y) \land \mathsf{R}(x, z) \land \mathsf{R}(y, z)$.
3. $\exists x, y, z.\ \mathsf{R}(x, y) \land \mathsf{R}(x, z) \land \mathsf{R}(y, z) \land \mathsf{R}(x, x)$.
4. $\exists v, w.\ \mathsf{S}(x, a, y) \land \mathsf{S}(x, v, y) \land \mathsf{S}(x, w, y) \land \mathsf{S}(x, x, x)$.

Solution.

1. $\exists x, y.\ \mathsf{R}(x, y)$.
2. $\exists x, y, z.\ \mathsf{R}(x, y) \land \mathsf{R}(x, z) \land \mathsf{R}(y, z)$.
3. $\exists x.\ \mathsf{R}(x, x)$.
4. $\exists v, w.\ \mathsf{S}(x, a, y) \land \mathsf{S}(x, x, x)$.

Exercise 5

Exercise. Consider a fixed set of relation names $\mathcal{R} = \{ \mathsf{R}_1, \ldots, \mathsf{R}_n \}$, each with a given arity $ar(\mathsf{R}_i)$.

1. Show that there is a BCQ q_{min} without constant symbols that is most specific, i.e., such that for any BCQ q without constant symbols, we have $q_{\text{min}} \sqsubseteq q$.
2. Is there also a most general BCQ q_{max} that contains all BCQs without constant names?
3. What if the considered BCQs may use constant names?
4. What if we consider FO queries instead?

Exercise 5

Exercise. Consider a fixed set of relation names $\mathcal{R} = \{ \mathsf{R}_1, \ldots, \mathsf{R}_n \}$, each with a given arity $ar(\mathsf{R}_i)$.

1. Show that there is a BCQ q_{min} without constant symbols that is most specific, i.e., such that for any BCQ q without constant symbols, we have $q_{\mathsf{min}} \sqsubseteq q$.
2. Is there also a most general BCQ q_{max} that contains all BCQs without constant names?
3. What if the considered BCQs may use constant names?
4. What if we consider FO queries instead?

Solution.

Exercise 5

Exercise. Consider a fixed set of relation names $\mathcal{R} = \{ \mathsf{R}_1, \ldots, \mathsf{R}_n \}$, each with a given arity $ar(\mathsf{R}_i)$.

1. Show that there is a BCQ q_{min} without constant symbols that is most specific, i.e., such that for any BCQ q without constant symbols, we have $q_{\text{min}} \sqsubseteq q$.
2. Is there also a most general BCQ q_{max} that contains all BCQs without constant names?
3. What if the considered BCQs may use constant names?
4. What if we consider FO queries instead?

Solution.

1. $q_{\text{min}} = \exists x.\, \mathsf{R}_1(x, \ldots, x) \wedge \cdots \wedge \mathsf{R}_n(x, \ldots, x)$.

Exercise 5

Exercise. Consider a fixed set of relation names $\mathcal{R} = \{ \mathsf{R}_1, \ldots, \mathsf{R}_n \}$, each with a given arity $ar(\mathsf{R}_i)$.

1. Show that there is a BCQ q_{min} without constant symbols that is most specific, i.e., such that for any BCQ q without constant symbols, we have $q_{\text{min}} \sqsubseteq q$.
2. Is there also a most general BCQ q_{max} that contains all BCQs without constant names?
3. What if the considered BCQs may use constant names?
4. What if we consider FO queries instead?

Solution.

1. $q_{\text{min}} = \exists x.\, \mathsf{R}_1(x, \ldots, x) \wedge \cdots \wedge \mathsf{R}_n(x, \ldots, x)$.
2. - Assume that some $q_{\text{max}} = \exists \mathbf{x}.\, \mathsf{R}_{i_1}(x_1^{i_1}, \ldots, x_{ar(\mathsf{R}_{i_1})}^{i_1}) \wedge \cdots \wedge \mathsf{R}_{i_\ell}(x_1^{i_\ell}, \ldots, x_{ar(\mathsf{R}_{i_\ell})}^{i_\ell})$ is indeed maximal.

Exercise 5

Exercise. Consider a fixed set of relation names $\mathcal{R} = \{ \mathsf{R}_1, \ldots, \mathsf{R}_n \}$, each with a given arity $ar(\mathsf{R}_i)$.

1. Show that there is a BCQ q_{min} without constant symbols that is most specific, i.e., such that for any BCQ q without constant symbols, we have $q_{\mathrm{min}} \sqsubseteq q$.
2. Is there also a most general BCQ q_{max} that contains all BCQs without constant names?
3. What if the considered BCQs may use constant names?
4. What if we consider FO queries instead?

Solution.

1. $q_{\mathrm{min}} = \exists x.\, \mathsf{R}_1(x, \ldots, x) \wedge \cdots \wedge \mathsf{R}_n(x, \ldots, x)$.
2.
   - Assume that some $q_{\mathrm{max}} = \exists \mathbf{x}.\, \mathsf{R}_{i_1}(x_1^{i_1}, \ldots, x_{ar(\mathsf{R}_{i_1})}^{i_1}) \wedge \cdots \wedge \mathsf{R}_{i_\ell}(x_1^{i_\ell}, \ldots, x_{ar(\mathsf{R}_{i_\ell})}^{i_\ell})$ is indeed maximal.
   - Then $\mathsf{R}_{i_j} \sqsubseteq q_{\mathrm{max}}$, and hence $\mathsf{R}_{i_j} \equiv q_{\mathrm{max}}$ for all $1 \leq j \leq \ell$.

Exercise 5

Exercise. Consider a fixed set of relation names $\mathcal{R} = \{ R_1, \ldots, R_n \}$, each with a given arity $ar(R_i)$.

1. Show that there is a BCQ $q_{\min}$ without constant symbols that is most specific, i.e., such that for any BCQ q without constant symbols, we have $q_{\min} \sqsubseteq q$.
2. Is there also a most general BCQ $q_{\max}$ that contains all BCQs without constant names?
3. What if the considered BCQs may use constant names?
4. What if we consider FO queries instead?

Solution.

1. $q_{\min} = \exists x.\, R_1(x, \ldots, x) \wedge \cdots \wedge R_n(x, \ldots, x)$.
2. - Assume that some $q_{\max} = \exists \mathbf{x}.\, R_{i_1}(x_1^{i_1}, \ldots, x_{ar(R_{i_1})}^{i_1}) \wedge \cdots \wedge R_{i_\ell}(x_1^{i_\ell}, \ldots, x_{ar(R_{i_\ell})}^{i_\ell})$ is indeed maximal.
   - Then $R_{i_j} \sqsubseteq q_{\max}$, and hence $R_{i_j} \equiv q_{\max}$ for all $1 \leq j \leq \ell$.
   - Therefore, unless $n = 1$, no such $q_{\max}$ exists.

Exercise 5

Exercise. Consider a fixed set of relation names $\mathcal{R} = \{\mathsf{R}_1, \ldots, \mathsf{R}_n\}$, each with a given arity $ar(\mathsf{R}_i)$.

1. Show that there is a BCQ $q_{\min}$ without constant symbols that is most specific, i.e., such that for any BCQ q without constant symbols, we have $q_{\min} \sqsubseteq q$.
2. Is there also a most general BCQ $q_{\max}$ that contains all BCQs without constant names?
3. What if the considered BCQs may use constant names?
4. What if we consider FO queries instead?

Solution.

1. $q_{\min} = \exists x.\, \mathsf{R}_1(x, \ldots, x) \wedge \cdots \wedge \mathsf{R}_n(x, \ldots, x)$.
2. - Assume that some $q_{\max} = \exists \mathbf{x}.\, \mathsf{R}_{i_1}(x_1^{i_1}, \ldots, x_{ar(\mathsf{R}_{i_1})}^{i_1}) \wedge \cdots \wedge \mathsf{R}_{i_\ell}(x_1^{i_\ell}, \ldots, x_{ar(\mathsf{R}_{i_\ell})}^{i_\ell})$ is indeed maximal.
   - Then $\mathsf{R}_{i_j} \sqsubseteq q_{\max}$, and hence $\mathsf{R}_{i_j} \equiv q_{\max}$ for all $1 \leq j \leq \ell$.
   - Therefore, unless $n = 1$, no such $q_{\max}$ exists.
3. $q_{\min}$ is a conjunction of every fact in the database instance, and $q_{\max}$ doesn't exist in general.

Exercise 5

Exercise. Consider a fixed set of relation names $\mathcal{R} = \{ \mathsf{R}_1, \ldots, \mathsf{R}_n \}$, each with a given arity $ar(\mathsf{R}_i)$.

1. Show that there is a BCQ q_{min} without constant symbols that is most specific, i.e., such that for any BCQ q without constant symbols, we have $q_{\mathsf{min}} \sqsubseteq q$.
2. Is there also a most general BCQ q_{max} that contains all BCQs without constant names?
3. What if the considered BCQs may use constant names?
4. What if we consider FO queries instead?

Solution.

1. $q_{\mathsf{min}} = \exists x.\ \mathsf{R}_1(x, \ldots, x) \land \cdots \land \mathsf{R}_n(x, \ldots, x)$.
2.
   - Assume that some $q_{\mathsf{max}} = \exists \mathbf{x}.\ \mathsf{R}_{i_1}(x_1^{i_1}, \ldots, x_{ar(\mathsf{R}_{i_1})}^{i_1}) \land \cdots \land \mathsf{R}_{i_\ell}(x_1^{i_\ell}, \ldots, x_{ar(\mathsf{R}_{i_\ell})}^{i_\ell})$ is indeed maximal.
   - Then $\mathsf{R}_{i_j} \sqsubseteq q_{\mathsf{max}}$, and hence $\mathsf{R}_{i_j} \equiv q_{\mathsf{max}}$ for all $1 \leq j \leq \ell$.
   - Therefore, unless $n = 1$, no such q_{max} exists.
3. q_{min} is a conjunction of every fact in the database instance, and q_{max} doesn't exist in general.
4. We could set $q_{\mathsf{min}} = \bot$, and $q_{\mathsf{max}} = \top$.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

1. - Suppose that the algorithm terminates with non-minimal q' for input CQ q.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

1. - Suppose that the algorithm terminates with non-minimal q' for input CQ q.
   - Then there is some atom $\mathsf{R}(\mathbf{x})$ in q' that was kept, but is redundant; let q'' be q' without this atom.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

1.
   - Suppose that the algorithm terminates with non-minimal q' for input CQ q.
   - Then there is some atom $\mathsf{R}(\mathbf{x})$ in q' that was kept, but is redundant; let q'' be q' without this atom.
   - Then $q'' \equiv q' \equiv q$; in particular, there is a homomorphism φ from q to q''.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

1.
   - Suppose that the algorithm terminates with non-minimal q' for input CQ q.
   - Then there is some atom $\mathsf{R}(\mathbf{x})$ in q' that was kept, but is redundant; let q'' be q' without this atom.
   - Then $q'' \equiv q' \equiv q$; in particular, there is a homomorphism φ from q to q''.
   - Let q''' be q without the atom $\mathsf{R}(\mathbf{x})$. Then there is a homomorphism ψ from q'' to q'''.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

1.
    - Suppose that the algorithm terminates with non-minimal q' for input CQ q.
    - Then there is some atom $\mathsf{R}(\mathbf{x})$ in q' that was kept, but is redundant; let q'' be q' without this atom.
    - Then $q'' \equiv q' \equiv q$; in particular, there is a homomorphism φ from q to q''.
    - Let q''' be q without the atom $\mathsf{R}(\mathbf{x})$. Then there is a homomorphism ψ from q'' to q'''.
    - But then $\psi \circ \varphi$ is a homomorphism from q to q''', so $q''' \sqsubseteq q$. Contradiction, since $\mathsf{R}(\mathbf{x})$ was kept.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

1. - Suppose that the algorithm terminates with non-minimal q' for input CQ q.
   - Then there is some atom $\mathrm{R}(\mathbf{x})$ in q' that was kept, but is redundant; let q'' be q' without this atom.
   - Then $q'' \equiv q' \equiv q$; in particular, there is a homomorphism φ from q to q''.
   - Let q''' be q without the atom $\mathrm{R}(\mathbf{x})$. Then there is a homomorphism ψ from q'' to q'''.
   - But then $\psi \circ \varphi$ is a homomorphism from q to q''', so $q''' \sqsubseteq q$. Contradiction, since $\mathrm{R}(\mathbf{x})$ was kept.
2. - Suppose that q_1, q_2 are cores of a CQ q.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

1. - Suppose that the algorithm terminates with non-minimal q' for input CQ q.
   - Then there is some atom $\mathsf{R}(\mathbf{x})$ in q' that was kept, but is redundant; let q'' be q' without this atom.
   - Then $q'' \equiv q' \equiv q$; in particular, there is a homomorphism φ from q to q''.
   - Let q''' be q without the atom $\mathsf{R}(\mathbf{x})$. Then there is a homomorphism ψ from q'' to q'''.
   - But then $\psi \circ \varphi$ is a homomorphism from q to q''', so $q''' \sqsubseteq q$. Contradiction, since $\mathsf{R}(\mathbf{x})$ was kept.
2. - Suppose that q_1, q_2 are cores of a CQ q.
   - Then $q_1 \equiv q \equiv q_2$.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

1.
   - Suppose that the algorithm terminates with non-minimal q' for input CQ q.
   - Then there is some atom $\mathrm{R}(\mathbf{x})$ in q' that was kept, but is redundant; let q'' be q' without this atom.
   - Then $q'' \equiv q' \equiv q$; in particular, there is a homomorphism φ from q to q''.
   - Let q''' be q without the atom $\mathrm{R}(\mathbf{x})$. Then there is a homomorphism ψ from q'' to q'''.
   - But then $\psi \circ \varphi$ is a homomorphism from q to q''', so $q''' \sqsubseteq q$. Contradiction, since $\mathrm{R}(\mathbf{x})$ was kept.
2.
   - Suppose that q_1, q_2 are cores of a CQ q.
   - Then $q_1 \equiv q \equiv q_2$.
   - Hence, there are homomorphisms φ_1 from q to q_1 and φ_2 from q to q_2.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

1.
   - Suppose that the algorithm terminates with non-minimal q' for input CQ q.
   - Then there is some atom $\mathsf{R}(\mathbf{x})$ in q' that was kept, but is redundant; let q'' be q' without this atom.
   - Then $q'' \equiv q' \equiv q$; in particular, there is a homomorphism φ from q to q''.
   - Let q''' be q without the atom $\mathsf{R}(\mathbf{x})$. Then there is a homomorphism ψ from q'' to q'''.
   - But then $\psi \circ \varphi$ is a homomorphism from q to q''', so $q''' \sqsubseteq q$. Contradiction, since $\mathsf{R}(\mathbf{x})$ was kept.
2.
   - Suppose that q_1, q_2 are cores of a CQ q.
   - Then $q_1 \equiv q \equiv q_2$.
   - Hence, there are homomorphisms φ_1 from q to q_1 and φ_2 from q to q_2.
   - Let ψ_1 be the restriction of φ_1 to q_2, and ψ_2 be the restriction of φ_2 to q_1.

Exercise 6

Exercise. Explain why the CQ minimisation algorithm is correct:

1. Why is the result guaranteed to be a minimal CQ?
2. Why is the result guaranteed to be unique up to bijective renaming of variables?

Definition (Lecture 10, Slide 10)

A conjunctive query q is *minimal* if:

- for all subqueries q' of q (that is, queries q' that are obtained by dropping one or more atoms from q),
- we find that $q' \not\equiv q$.

A minimal CQ is also called a *core*.

Solution.

1.
   - Suppose that the algorithm terminates with non-minimal q' for input CQ q.
   - Then there is some atom $\mathrm{R}(\mathbf{x})$ in q' that was kept, but is redundant; let q'' be q' without this atom.
   - Then $q'' \equiv q' \equiv q$; in particular, there is a homomorphism φ from q to q''.
   - Let q''' be q without the atom $\mathrm{R}(\mathbf{x})$. Then there is a homomorphism ψ from q'' to q'''.
   - But then $\psi \circ \varphi$ is a homomorphism from q to q''', so $q''' \sqsubseteq q$. Contradiction, since $\mathrm{R}(\mathbf{x})$ was kept.
2.
   - Suppose that q_1, q_2 are cores of a CQ q.
   - Then $q_1 \equiv q \equiv q_2$.
   - Hence, there are homomorphisms φ_1 from q to q_1 and φ_2 from q to q_2.
   - Let ψ_1 be the restriction of φ_1 to q_2, and ψ_2 be the restriction of φ_2 to q_1.
   - Then ψ_1 and ψ_2 are surjective, so q_1 and q_2 must be isomorphic.