Friendly Logics, Fall 2005 (partial lecture notes)

Val Tannen

1 Preliminaries

Do you know the answer to the following questions?

- What is a language of strings (words) over an alphabet?
- What is an r.e. language?
- What is a computable function?
- What is a decidable language?
- What is a co-r.e. language? How do r.e. and co-r.e. languages relate to decidable languages?
- Is the Halting Problem r.e.? Is it decidable?
- Can you think of an undecidable language that is not defined in terms of Turing Machines?
- What is a many-one reduction?
- What is an r.e.-complete (or co-r.e.-complete) language?
- What is a first-order (FO) vocabulary?
- What is an FO formula? What is an FO sentence?
- What is an FO structure (also called model, interpretation, or instance)?
- If $\mathcal{A}$ is a structure and φ is a sentence, what does $\mathcal{A} \models \varphi$ (read $\mathcal{A}$ is a model of φ) mean?
- If Γ is a set of sentences and φ is a sentence, what does $\Gamma \models \varphi$ (read φ is a logical consequence of Γ) mean?
- What is a valid sentence?
- What is a satisfiable sentence?
- What is an FO proof?
- If Γ is a set of sentences and φ is a sentence, what does $\Gamma \vdash \varphi$ (read φ is provable from Γ) mean?

If you don't, review computability from your favorite text and/or logic from a good book such as:

"Logic for Computer Science", J. Gallier, (Wiley 1986), out of print but available online with the latest revisions at `http://www.cis.upenn.edu/~jean/gbooks/logic.html`

"A mathematical introduction to logic", H. B. Enderton, Harcourt/Academic Press, 1972 and 2000.

"Introduction to mathematical logic", E. Mendelson, various publishers, 1964, 1979, 1987, and 1997.

"Computability and logic", G. Boolos and R. Jeffrey, Cambridge Univ. Press, 1974 and 1980.

2 Unfriendly Aspects of First-Order Logic, in General

Our starting point is the semantics of first-order logic(FOL), specifically **logical consequence** assertions. These have the form $\Gamma \models \varphi$ where φ is an FO sentence and Γ is set of FO sentences. For example, we can think of Γ as an "axiomatization" of a mathematical theory or as part of a "modelization" of a physical or informatic system. So our initial question is: what kind of computing can we do with logical consequence assertions?

Some theories are axiomatized by finite sets of axioms but most make use of so-called *axiom schemes*, which are finite descriptions of infinite sets of axioms. Thus, we will consider so-called *recursively axiomatized* theories, given by a Γ that is decidable. [1]

Starting with the work of Frege, several equivalent **proof systems** for FOL were formalized by Hilbert and others. Answering a question of Hilbert, Gödel showed that the intuition captured by the proof systems was correct:

Theorem 2.1 (Gödel's Completeness Theorem) $\Gamma \models \varphi$ *iff* $\Gamma \vdash \varphi$.

The FOL proof systems and, in fact, any reasonable proof system has a nice computational nature: proofs are finite objects, it is decidable whether a finite object is a correct proof and the sentence proven by a proof can be computed from it. This implies that $\{(\Gamma, \varphi) \mid \Gamma \vdash \varphi\}$ is r.e. By Gödel's Completeness Theorem and defining

$$VALID \stackrel{\text{def}}{=} \{\varphi \mid \models \varphi\}$$

we have in particular:

Corollary 2.2 *VALID is r.e.*

This is good, but even better would be if we could actually *decide* first-order provability (hence logical consequence). This question, known as the *Entscheidungsproblem* [2], and again asked by Hilbert, has a negative answer:

[1] Note that when Γ is finite $\Gamma \models \varphi$ is equivalent to $\models \Gamma \rightarrow \varphi$ (interpreting Γ as the conjunction of its elements).

[2] "Decision problem" in German. At the beginning of the 20^{th} century most papers on mathematical logic were in German.

Theorem 2.3 (Church/Turing's Undecidability Theorem) *VALID is undecidable.*

We still wish to find cases in which logical consequence is decidable. and there are two strategies we use in what follows: to restrict ourselves to FO sentences of a special form and to restrict ourselves to special classes of models.

An obvious restriction of the second kind is to look at validity over **finite models**. In fact, in computer science we are concerned mostly with finite structures so this is very natural. Define

$$\textit{FIN-VALID} \stackrel{\text{def}}{=} \{\varphi \mid \mathcal{A} \models \varphi \text{ for all finite } \mathcal{A}\}$$

Exercise 2.1 *Give an example of an FO sentence that is finitely valid but not valid.*

Unfortunately, finite validity is also unfriendly! We have:

Theorem 2.4 (Trakhtenbrot's Theorem) *FIN-VALID is undecidable.*

In fact, *FIN-VALID* is not even r.e.! This is easy to see if we look at the **satisfiability** property.

Recall that a sentence φ is (*finitely*) *satisfiable* if there exists a (finite) model $\mathcal{A}$ such that $\mathcal{A} \models \varphi$. Note that φ is satisfiable iff $\neg\varphi$ is invalid and φ is valid iff $\neg\varphi$ is unsatisfiable. Hence, it follows from the Church/Turing and Trakhtenbrot theorems that satisfiability and finite satisfiability are both undecidable. Moreover, it follows from Gödel's Completeness Theorem that

Corollary 2.5 *The set of satisfiable FO sentences is co-r.e.*

The situation with finite satisfiability is quite different:

Proposition 2.6 *The set of finitely satisfiable FO sentences is r.e.*

Proof Isomorphic models satisfy the same sentences so it's sufficient to consider finite models whose universe (domain) is of the form $\{1, 2, \ldots, n\}$. We can enumerate all pairs consisting of such a finite model and an FO sentence, and output the sentence if it holds in the model. (This assumes that for finite models $\mathcal{A} \models \varphi$ is decidable; see Theorem 6.2.) □

Note that although finite satisfiability is r.e., it is nonetheless undecidable, as follows from Trakhtenbrot's Theorem. Moreover,

Corollary 2.7 *FIN-VALID is not r.e.*

Proof If *FIN-VALID* was r.e. then finite satisfiability would be co-r.e., hence also decidable, by Proposition 2.6. This would make *FIN-VALID* decidable which contradicts Trakhtenbrot's Theorem. □

Therefore, FOL " in finite models" is worse than standard FOL: it does not even admit a complete proof system!

3 The Church-Turing and Trakhtenbrot Theorems

Here is a proof of Theorem 2.3.

Recall the String Rewriting Problem in my [[**lecture notes on computability**]]. A many-one reduction $K \leq SRP$ is given there, showing that SRP is undecidable. $\Leftarrow$

Here we show that $SRP \leq VALID$.

We will describe the total computable function that performs the reduction $SRP \leq VALID$ by the FC-program that computes it. This program will take an input of the form $\langle R, u, v\rangle$ where R is a finite set of string rewrite rules over an alphabet Σ and u, v are strings over Σ and produce an output of the form ϕ. In order for this to be a many-one reduction, we need $u \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} v$ iff $\models \phi$.

First, the reduction will have to construct the first-order language over which ϕ is built. This will contain a binary a binary predicate symbol, Rew. Moreover, let Σ be the alphabet over which the strings in R and u, v are built. For each $\sigma \in \Sigma$, the first-order language will contain a unary function symbol f_σ. Finally, we have a constant symbol c. From each rewrite rule $r \in R$,

$$r \ : \ \sigma_1 \cdots \sigma_m \longrightarrow \tau_1 \cdots \tau_n$$

the reduction will construct a sentence

$$\Phi_r \overset{\text{def}}{=} \forall x \ Rew(f_{\sigma_1}(\cdots f_{\sigma_m}(x)\cdots)\,,\, f_{\tau_1}(\cdots f_{\tau_n}(x)\cdots))$$

and then, if $R = \{r_1, \ldots, r_k\}$, the reduction will construct

$$\Phi_R \overset{\text{def}}{=} \Phi_{r_1} \wedge \cdots \wedge \Phi_{r_k}$$

Moreover, if $u \equiv \delta_1 \cdots \delta_p$ and $v \equiv \varepsilon_1 \cdots \varepsilon_q$, the reduction will construct

$$\Phi_{u,v} \overset{\text{def}}{=} Rew(f_{\sigma_1}(\cdots f_{\sigma_m}(c)\cdots)\,,\, f_{\tau_1}(\cdots f_{\tau_n}(c)\cdots))$$

We also need to say something that ensures that the meaning of the predicate symbol Rew always "simulates" a rewriting relation. Taking

$$\Phi_{Rew} \overset{\text{def}}{=} (\forall x \ Rew(x,x)) \ \wedge \ (\forall x,y,z \ Rew(x,y) \wedge Rew(y,z) \rightarrow Rew(x,z)) \wedge$$
$$\wedge \bigwedge_{\sigma\in\Sigma} (\forall x,y \ Rew(x,y) \rightarrow Rew(f_\sigma(x), f_\sigma(y)))$$

the reduction will finally construct

$$\phi \overset{\text{def}}{=} \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{u,v}$$

This is clearly defining a total computable function. It remains to show that $u \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} v$ iff $\models \phi$.

Claim 1. $u \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} v \ \supset \models \phi$

Proof of Claim 1. By soundness, it is sufficient to show $u \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} v \ \supset \ \vdash \phi$. This is shown by induction on the length of the rewriting sequence from u to v. More precisely, one can show that

- for any $w \in \Sigma^*$, $\quad \vdash \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{w,w}$
- for any $r \in R, w_1, w_2 \in \Sigma^*$, if $w_1 \xrightarrow{r} w_2$ then $\quad \vdash \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{w_1,w_2}$, and
- for any $w_1, w_2, w_3 \in \Sigma^*$, if $\vdash \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{w_1,w_2}$ and $\vdash \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{w_2,w_3}$ then $\vdash \Phi_R \wedge \Phi_{Rew} \rightarrow \Phi_{w_1,w_3}$.

The details are omitted. **End of Proof of Claim 1.**

Claim 2. $\models \phi \supset u \xrightarrow{R}\!\!\!\!\!\rightarrow v$

Proof of Claim 2. Since $\models \phi$ then $\mathcal{R} \models \phi$ where $\mathcal{R}$ is the following structure: the underlying set is Σ^*, *Rew* is interpreted as the rewrite relation $\xrightarrow{R}\!\!\!\!\!\rightarrow$, f_σ is interpreted as the function that concatenates σ at the beginning of its argument, and c is interpreted as nil, the empty string. Clearly $\mathcal{R} \models \Phi_R \wedge \Phi_{Rew}$. Since we also have $\mathcal{R} \models \phi$, it follows that $\mathcal{R} \models \Phi_{u,v}$ hence $u \xrightarrow{R}\!\!\!\!\!\rightarrow v$. **End of Proof of Claim 2.**

End of proof.

Note that the first-order language needs to contain at least one binary predicate symbol, one constant symbol, and a number of unary function symbols for the previous proof to go through. It turns out that the number of function symbols can be reduced to two, since the String Rewrite Problem over the alphabet $\{0, 1\}$ is undecidable (to see this, recall that strings over an arbitrary alphabet can be encoded as strings of bits).

We now turn to Trakhtenbrot's Theorem (Theorem 2.4). In [[**Libkin's**]] "Elements of Finite Model Theory", pp 165–168, it is shown how to compute from the description of a Turing Machine M a FO sentence φ_M such that M halts on the empty string input iff φ_M is finitely satisfiable. Like K, K_ϵ is undecidable, (in fact, it is r.e.-complete, just like K). It follows that ⇐

$$\textit{FIN-SAT} \stackrel{\text{def}}{=} \{\varphi \mid \text{ there exists a finite } \mathcal{A} \text{ such that } \mathcal{A} \models \varphi\}$$

is undecidable. Using the reduction $\varphi \longmapsto \neg\varphi$ we conclude that $\overline{\textit{FIN-VALID}}$ and therefore *FIN-VALID* are also undecidable.

Exercise 3.1 *Show how to modify the proof of Trakhtenbrot's Theorem in Libkin's book in order to prove the Church-Turing Theorem.*

Exercise 3.2 *VALID and FIN-SAT are both r.e.-complete hence many-one reducible to each other. Describe as best you can two total computable functions that realize these two many-one reductions.*

4 Reduction Classes

Let us now look at the other strategy for finding decidable cases of logical consequence: restricting the class of sentences.

Unfortunately, we begin with a disappointment: even for (apparently) simple classes of sentences validity is undecidable.

Definition 4.1 A class C of FO sentences is called a **reduction class** if there is a computable function f that maps arbitrary FO sentences into C-sentences such that φ is valid iff $f(\varphi)$ is valid. It follows from the Church/Turing Theorem that the validity problem for C is also undecidable.

Intuitively, reduction classes have a validity problem that is "as hard as" the validity problem for all of FOL. Even before the Church/Turing result, various reduction classes were exhibited (note that you don't really need a formalized definition of computability for such results). We give an example in what follows.

Definition 4.2 A formula of the form $\varphi \stackrel{\text{def}}{=} Q_1 x_1 \cdots Q_n x_n \; \psi$ where each Q_i is either $\forall$ or $\exists$ and ψ is quantifier-free is called a *prenex* formula. If the quantifiers are all $\forall$ (all $\exists$) then φ is called a *universal* formula (an *existential* formula).

In what follows, by **equivalent** sentences we mean two sentences that are logical consequences of each other. This is denoted $\varphi \models\!\!\models \psi$ and it holds iff $\models \varphi \leftrightarrow \psi$.

Lemma 4.1 *For each sentence we can compute (in PTIME) an equivalent prenex sentence.*

Proof "Pull out" the quantifiers by repeatedly using transformations like

$$\neg\exists x\, \varphi \longmapsto \forall x\, \neg\varphi \qquad \varphi \wedge (\forall x\, \psi) \longmapsto \forall x(\varphi \wedge \psi) \qquad (\forall x\, \varphi) \rightarrow \psi \longmapsto \exists x(\varphi \rightarrow \psi)$$

etc., while renaming bound variables to avoid unintended scope capture. □

Exercise 4.1 *Analyze the complexity of the algorithm sketched above; for any FO sentence it computes an equivalent prenex FO sentence. You can choose the data structure used to represent sentences.*

Lemma 4.2 (Skolem) *Let $\mathcal{V}$ be a vocabulary and let $\overline{\mathcal{V}}$ be its extension with countably many fresh function symbols of each arity (including nullary functions i.e. constants). For each prenex sentence φ over $\mathcal{V}$ we can compute (in PTIME) a sentence $Sk(\varphi)$ over $\overline{\mathcal{V}}$ such that*

- *$Sk(\varphi)$ is a universal sentence.*
- *φ is true in the $\mathcal{V}$-restriction of any model of $Sk(\varphi)$.*
- *Any model of φ can be extended, keeping the same universe (domain), to a $\overline{\mathcal{V}}$-structure that satisfies $Sk(\varphi)$. (Hence, φ is satisfiable iff $Sk(\varphi)$ is satisfiable.)*

Proof Eliminate the existential quantifiers from left to right in the prenex sentence repeating the transformation

$$\forall x_1 \cdots \forall x_n \exists y\, \varphi(y) \longmapsto \forall x_1 \cdots \forall x_n\, \varphi(f(x_1, \ldots, x_n))$$

where f is a fresh functions symbol.

For example:

$$Sk(\exists u\,\forall x\,\exists v\,\forall y\,\exists w\,R(u,v)\wedge f(x,w)=g(v)) \stackrel{\text{def}}{=} \forall x\,\forall y\,R(r,s(x))\wedge f(x,t(x,y))=g(s(x))$$

□

$Sk(\varphi)$ is unique up to some renaming so it is called the *Skolem Normal Form* of φ. The transformation $\varphi \longmapsto Sk(\varphi)$ is called *skolemization*. Symbols like r, s and t in the proof example are called *Skolem functions*.

Exercise 4.2 *Analyze the complexity of the algorithm sketched above; for any prenex FO sentence it computes an equivalent Skolem normal form FO sentence. Again, you can choose the data structure used to represent sentences.*

Theorem 4.3 *The existential sentences form a reduction class.*

Proof Given an FO sentence φ and assuming that the transformations to prenex form are implicit, observe that $\neg Sk(\neg\varphi)$ is an existential sentence that is valid iff φ is valid. □

Warning! Our definition of reduction class is for the *validity* decision problem. It is more common to define reduction classes for the *satisfiability* decision problem. If the class is closed under negation the two coincide. But classes defined by the quantifier structure of prenex formula are typically *not* closed under negation! For example, the theorem above is often stated as "the universal sentences form a reduction class (wrt decidability of satisfiability)".

5 The Finite Model Property

Although finite validity has bad computational properties for the class of *all* FO sentences, the r.e.-ness of finite satisfiability can be exploited for classes of restricted FO sentences.

Definition 5.1 A class of sentences has the **finite model property** if any satisfiable sentence in the class is also finitely satisfiable.

The class of all FO sentences does not have the finite model property. Indeed, it is fairly easy to concoct sentences that are satisfiable but not finitely satisfiable . Such sentences are called *infinity axioms*. In fact, if the class of all FO sentences would have the finite model property then the next result would contradict the undecidability results shown earlier!

Proposition 5.1 *Let C be a class of sentences that is decidable (i.e., it is decidable whether an FO sentence φ is in C). If C has the finite model property then satisfiability of C-sentences is decidable.*

Proof Recall that for the class of all FO sentences satisfiability is co-r.e. and finite satisfiability is r.e. Since C is decidable it enjoys the same properties. But for the sentences in C satisfiability and finite satisfiability coincide! Hence they are both r.e. and co-r.e. and thus decidable. □

The decision procedure provided by the previous proof is very inconvenient: it consists of trying, in parallel, to finitely satisfy the sentence and to prove its negation (if the sentence is satisfiable then the first thread succeeds; if not then the second thread does). A better procedure is given by the following.

Definition 5.2 A class of sentences has the **small model property** if there is a total recursive function u such that any satisfiable sentence in the class has a model with less than $u(|\ \varphi\ |)$ elements. (Here $|\ \varphi\ |$ denotes the **size** of φ.)

The small model property implies the finite model property. However, a decidable class of sentences with the small model property has a decision procedure for satisfiability that is potentially simpler. There is no need for the annoying attempt to prove the negation of a sentence φ; it suffices to check all models with less than $u(|\ \varphi\ |)$ elements.

Note that we only said *potentially* simpler: it all depends on how easy to compute u is. Indeed, if C is decidable then the finite model property implies the small model property! Consider the following algorithm for u:

> On input n, generate the (finitely many) sentences in C of size n. For each of them, in parallel, check for finite satisfiabilty and try to prove the negation, and thus compute either the size of a model or 0 (if unsatisfiable). Return as $u(n)$ the largest of these.

Using this u the small model property gives a decision procedure that is just as incovenient as the one given by the finite model property. If we consider classes C that are not decidable a general observation is that there exist such classes that have the finite model property but do not have the small model property: simply take the class of all finitely satisfiable sentences. Indeed, suppose there is a total recursive function u such that any satisfiable sentence φ in this class has a model with less than $u(|\ \varphi\ |)$ elements. But all the sentences in this class are satisfiable! Hence φ is finitely satisfiable iff it has a model with less than $u(|\ \varphi\ |)$ elements. This would make finite satisfiability decidable. [[**Thanks to Madhu and Scott**]]. ⇐

Anyway, what happens for concrete classes of sentences is that a better and more specific u is derived which moreover gives a useful complexity upper bound for the decision procedure. Here is an example:

Theorem 5.2 *The existential FO sentences have the small model property. In fact, any satisfiable existential sentence φ has a model with at most $|\ \varphi\ |$ elements.*

Proof Let $\varphi \stackrel{\text{def}}{=} \exists x_1 \cdots \exists x_m\, \psi$ be an existential sentence where ψ is quantifier-free. We replace each non-variable functional term occurring in ψ with a fresh existentially quantified variable and an additional equality atom. We do this bottom-up for all subterms, including constants, so if $t \equiv f(t_1, \ldots, t_k)$ is such a term and $t_1, \ldots, t_k$ are replaced by $y_1, \ldots, y_k$ then t is replaced by a fresh variable y and we add the equality atom $f(y_1, \ldots, y_k) = y$. Therefore, φ is transformed into a sentence $\overline{\varphi}$ of the form

$$\overline{\varphi} \stackrel{\text{def}}{=} \exists x_1 \cdots \exists x_m\, \exists y_1 \cdots \exists y_n\, \overline{\psi} \wedge f_1(\ldots) = y_1 \wedge \cdots \wedge f_n(\ldots) = y_n$$

where $f_1, \ldots, f_n$ are the functional symbol ocurrences in ψ (we treat constants as nullary functions).

For example, $\exists x\, R(x, f(c, x))$ is transformed into $\exists x\, \exists y_1 \exists y_2\, R(x, y_2) \wedge c = y_1 \wedge f(y_1, x) = y_2$.

It is not hard to see that φ and $\overline{\varphi}$ hold in the same models and that if $\overline{\varphi}$ is satisfiable then it has a model with at most $m + n$ elements. Since $n \leq \mid \psi \mid$ we conclude that if φ is satisfiable then it has a model with at most $\mid \varphi \mid$ elements. $\square$

Corollary 5.3 *Satisfiability of existential sentences is NP-complete.*

Proof For membership in NP, let the sentence be $\varphi \stackrel{\text{def}}{=} \exists x_1 \cdots \exists x_m\, \psi(x_1, \ldots, x_m)$. We guess a model $\mathcal{A}$ with less then $\mid \varphi \mid$ elements, we also guess a valuation v that maps each of $x_1, \ldots, x_m$ to some element of $\mathcal{A}$, and then we check $\mathcal{A}, v \models \psi$. Checking the truth of a quantifier-free formula can be done in PTIME in the size of the formula, see Theorem 6.1.

For NP-hardness we provide a reduction from boolean satisfiability. Given a boolean formula β with propositional variables $p_1, \ldots, p_n$ we construct the existential sentence $\varphi \stackrel{\text{def}}{=} \exists x_1 \cdots \exists x_n\, \overline{\beta}$ over a vocabulary with one unary predicate symbol R where $\overline{\beta}$ is obtained by replacing each p_i with $R(x_i)$. Now consider the "binary" model $\mathcal{B}$ whose universe is $\{0, 1\}$ and where R is interpreted as $\{1\}$. Clearly β is satisfiable iff $\mathcal{B} \models \varphi$. It is also easy to see that if φ is satisfiable then it holds true in $\mathcal{B}$. Therefore, β is satisfiable iff φ is. $\square$

And this finally gives a class of sentences for which the validity problem is decidable:

Corollary 5.4 *Validity of universal sentences is decidable and coNP-complete.*

This means that the satisfiability of *existential* sentences is also decidable and in fact NP-complete. (Note that Theorem 4.3 says that *validity* is undecidable for this same class.)

For certain simple existential sentences satisfiability is trivially decidable: they are in fact all satisfiable!

Exercise 5.1 *An* existential-conjunctive *sentence is a sentence of the form $\exists x_1 \cdots \exists x_n\, \varphi$ where φ is a conjunction of atomic formulas (equalities are allowed). Prove that any existential-conjunctive sentence is satisfiable in a model with one element.*

Such sentences are related to the *conjunctive queries* that we will study later.

6 Model Checking for FOL

Given a vocabulary $\mathcal{V}$, the **model checking** [3] problem consists of deciding if $\mathcal{A} \models \varphi$ for an FOL $\mathcal{V}$-sentence φ and a finite $\mathcal{V}$-structure $\mathcal{A}$.

We shall be interested in the complexity of the model checking problem, in fact in three different situations:

[3] This name comes from automated verification. We shall see later that model checking for many verification logics is a particular case of FOL model checking.

Combined complexity The input is $(\mathcal{A}, \varphi)$, its size is $\mid \mathcal{A} \mid + \mid \varphi \mid$.

Expression complexity $\mathcal{A}$ is fixed, the input is just φ.

Data complexity φ is fixed, the input is just $\mathcal{A}$.

We must be clear about what is meant by $\mid \mathcal{A} \mid$. This is the size of the complete description of $\mathcal{A}$. But how do we give such a description as input to a Turing machine? We fix an enumeration of the elements of the (non-empty) universe of $\mathcal{A}$: $a_1, a_2, \ldots a_n$ then we represent (encode) the model on Turing machine tapes with respect to this enumeration. Specifically, we represent a_k by the number k, in *binary*, therefore by using $\log k$ bits. We encode tuples and then relations using some separating symbols. The tuples in a relation will be ordered lexicographically, based on the ordering of domain elements given by the enumeration. For example, consider the enumerated domain a, b, c. Then $R = \{(c, b), (a, c), (a, b)\}$ $S = \{(c, a, c), (a, a, b)\}$ can be encoded by the string `R/0-1/0-10/10-1/S/0-0-1/10-0-10/`. The interpretation of function symbols of arity m is represented as $m + 1$-relations in which the tuples group the arguments and the corresponding result of the function. Finally, we make the size of the universe, in *unary* (to avoid pathologies) part of the representation.

If the universe of $\mathcal{A}$ has n elements and if the vocabulary contains an m-ary relation symbol R then the model description contains the encoding of as many as n^m tuples. In general, for a fixed vocabulary $\mid \mathcal{A} \mid$ is no more than polynomial in the size of the universe of $\mathcal{A}$.

We begin with with the quantifier-free case and it will be useful to consider a slightly more general problem, involving formulas and valuations rather than just sentences: decide if $\mathcal{A}, v \models \psi$ where ψ is a quantifier-free formula and v is a valuation defined for a set of variables that includes the free variables of ψ. In this case, we consider v, together with ψ, to be the input for expression complexity.

Theorem 6.1 *The data complexity of quantifier-free formula checking is in LOGSPACE. The expression and combined complexities of the same problem are in PTIME.*

Proof The procedure relies on evaluating the functional terms in the formula, determining the truth value of the atoms, and evaluating the resulting boolean expression. Term evaluation is done akin to arithmetic expression evaluation, using a stack. This gives us the PTIME expression and combined complexities. For the data complexity note that the size of the stacks does not depend on $\mid \mathcal{A} \mid$. We just need to lookup tuples in the description of $\mathcal{A}$ and this can be done with pointers of size $\log \mid \mathcal{A} \mid$. This gives the LOGSPACE bound on data complexity. □

Remark It has been shown that boolean expressions can be evaluated in LOGSPACE. (Indeed, to evaluate, e.g., $a \vee b$ we do not need to record both the value of a and b.) We would not be evaluating b at all if a resulted in `true`. Therefore, in the absence of function symbols, i.e., when the vocabulary consists only of relation symbols and constants, the expression and combined complexity of quantifier-free model checking are also in LOGSPACE [4]

[4]However, it seems that we cannot evaluate functional terms in LOGSPACE. (Think what goes wrong with the stack-based procedure.) It is known that there exist context-free languages that are NLOGSPACE-complete. I am still trying to find out whether such languages can be reduced to the problem of functional term evaluation.

Theorem 6.2 *The data complexity of FOL model checking is in LOGSPACE. The expression and combined complexities of FOL model checking are PSPACE-complete.*

Proof First we put the sentence in prenex form [5]. Let $Q_1x_1 \cdots Q_kx_k\ \psi$ be the result.

We evaluate this sentence in $\mathcal{A}$ using k nested loops iterating each of $x_1, \ldots, x_k$ through all the elements of the universe of $\mathcal{A}$. If x_i is universally quantified then the loop corresponding to x_i computes a big conjunction (disjunction if existentially quantified). In the innermost loop we evaluate $\mathcal{A}, v \models \psi$ where v is the valuation recording the current values of $x_1, \ldots, x_k$. If m is the number of elements of $\mathcal{A}$ then the time complexity of doing all this is

$$O(m^k\ poly(|\ \mathcal{A}\ |, |\ \psi\ |))$$

where *poly* is a two-variable polynomial. For the space complexity, note that we can keep track of the current v by using k pointers of size $\log m$. Thus the space complexity adds $O(k \log m)$ to the space complexity of the quantifier-free case. This gives LOGSPACE for data complexity and PSPACE for expression and combined complexities.

To show PSPACE-completeness we reduce from *QBF*, the problem of checking the truth of a (fully) quantified boolean formula. This is essentially the same reduction performed in the proof of Corollary 5.3.

Given a fully (no free propositional variables) quantified boolean formula $\gamma \stackrel{\text{def}}{=} Q_1p_1 \cdots Q_kp_k\ \beta$ we construct the FO sentence $\varphi \stackrel{\text{def}}{=} Q_1x_1 \cdots Q_kx_k\,\overline{\beta}$ over a vocabulary with one unary predicate symbol R where $\overline{\beta}$ is obtained by replacing each p_i with $R(x_i)$. Clearly γ is true iff $\mathcal{B} \models \varphi$ where $\mathcal{B}$ is defined in the proof of Corollary 5.3. Since $\mathcal{B}$ is a fixed model, this gives a lower bound for both combined complexity and expression complexity.

7 First-Order Queries

A *relational database schema* is a non-empty set Σ of relation symbols with their arities. Relational database formalisms also permit constants. In fact, we fix a countably infinite set $\mathbb{D}$ whose elements we call *constants*. These constants can appear in formulas, in other words we work with the FO vocabulary $\Sigma \cup \mathbb{D}$.

For semantics it is more covenient—although not essential—to give a treatment a little different than that of standard FOL. Namely the set $\mathbb{D}$ is also understood as the sole *universe of discourse* for the interpretation of formulas. A *relational database instance* for a given schema Σ (a Σ-*instance*) is a first-order structure whose domain, or universe, is $\mathbb{D}$ and in which the relation symbols are interpreted by *finite* relations, while the constants are interpreted as themselves.

For a given schema Σ , consider *queries* of the form $\{\mathbf{x} \mid \varphi\}$ where $\mathbf{x}$ is a tuple of variables or constants [6] and φ is a first-order formula *with equality* over Σ such that the free variables of φ

[5] I don't know if this can be done in LOGSPACE but for data complexity the sentence is fixed. It certainly can be done in low degree PTIME hence PSPACE.

[6] For these general FO queries, we could, without loss of generality, assume that the tuple $\mathbf{x}$ consists of distinct variables. This is not the case when we discuss conjunctive queries without equality atoms as we shall see later.

occur in $\mathbf{x}$. The *inputs* of the query are the Σ-instances. For each input $\mathcal{I}$, the *output* of the query $q \equiv \{\mathbf{x} \mid \varphi\}$ is the n-ary relation (where n is the length of $\mathbf{x}$)

$$q(\mathcal{I}) = \{v(\mathbf{x}) \mid \text{ valuation } v \text{ such that } \mathcal{I}, v \models \varphi\}.$$

Exercise 7.1 *We also allow the case when $n = 0$, that is $\mathbf{x}$ is the empty tuple, denoted $()$. What can the output of the query be in this case?*

But what is a *database*? Clearly, we expect it to be finite so we would only consider the case when the schema, Σ, is finite. The database consists of interpretations for the schema symbols, hence it is a finite collection of finite relations. A database cannot encompass the entire domain $\mathbb{D}$ which is infinite. Let $\mathcal{I}$ be an instance. The *active domain* of $\mathcal{I}$, notation $adom(\mathcal{I})$, is the set of all elements of $\mathbb{D}$ that actually appear in the interpretations in $\mathcal{I}$ for the relation symbols. While $\mathbb{D}$ is infinite, $adom(\mathcal{I})$ is always finite. Moreover, given a query $q \equiv \{\mathbf{x} \mid \varphi\}$, we will denote by $adom(q)$ the (finite) set of constants that occur in φ or $\mathbf{x}$.

It is our expectation that the database $\mathcal{I}$ together with the query q completely determine the output $q(\mathcal{I})$. In particular, only the elements in $adom(\mathcal{I}) \cup adom(q)$ can appear in the output. This is not the case for all FO queries. For example, the outputs of $\{x \mid \neg R(x)\}$ or $\{(x, y) \mid R(x) \vee S(y)\}$ are in fact infinite! More subtly, the following query is also problematic: $\{x \mid \forall y R(x, y)\}$. Here the output contains only elements from $adom(\mathcal{I})$ but whether a tuple is in the output or not depends on the set of elements we let y range over. These two queries are "dependent on the domain".

To capture this precisely, given a query $q \equiv \{\mathbf{x} \mid \varphi\}$, for any instance $\mathcal{I}$ and any D such that $adom(\mathcal{I}) \cup adom(q) \subseteq D \subseteq \mathbb{D}$, we denote by $q(\mathcal{I}/D)$ the output of the query on the input obtained by resticting the domain to D.

Definition 7.1 A query q is *domain independent* if for any $\mathcal{I}$ and any D_1, D_2 where $adom(\mathcal{I}) \cup adom(q) \subseteq D_i \subseteq \mathbb{D}, i = 1, 2$ we have $q(\mathcal{I}/D_1) = q(\mathcal{I}/D_2)$.

Exercise 7.2 *Give an example of a query q, an instance $\mathcal{I}$ and a D where $adom(\mathcal{I}) \cup adom(q) \subseteq D \subseteq \mathbb{D}$, such that $q(\mathcal{I}/D), q(\mathcal{I}/adom(\mathcal{I}) \cup adom(q))$ and $q(\mathcal{I}/\mathbb{D})$ are all* distinct.

It is generally agreed that in a reasonable *query language*, all the queries should be domain independent. Therefore, general first-order queries do not make a good query language. Worse

Theorem 7.1 *It is undecidable whether a first-order query is domain independent.*

Proof We reduce *FIN-VALID* to the problem of domain-independence and then the result follows from Trakhtenbrot's Theorem. The reduction maps

$$\varphi \quad \longmapsto \quad q \equiv \{x \mid \neg\varphi \wedge \neg R(x)\}$$

where R is a unary relation symbol that *does not occur* in φ. So $q's$ schema consists of the relation symbols in φ plus R. [7]

[7] As stated earlier, Trakhtenbrot's Theorem assumes a general FO vocabulary. It can be shown however that the class of *pure* (no function symbols) sentences is a reduction class for validity and, in fact, finite validity. It follows that finite validity of pure sentences is also undecidable.

If φ is finitely valid then q's output is always empty hence q is domain-idependent. If φ is not finitely valid then $\neg\varphi$ is true in some finite structure $\mathcal{A}$. We put the elements of the universe of $\mathcal{A}$ in one-to-one correspondence with some elements of $\mathbb{D}$ and consider a database instance $\mathcal{I}$ for the schema of q. In $\mathcal{I}$ the relation symbols in φ are interpreted according to the correspondence with $\mathcal{A}$ while R is interpreted as the empty set, hence $q(\mathcal{I}/D) = D$ for any $D \supseteq adom(\mathcal{I})$. It follows that q is not domain-independent. $\square$

So what do we do to get a reasonable query language? It is possible to define decidable *safety* restrictions on general first-order formulas such that the safe queries are domain independent and moreover for any domain independent query there exists an equivalent safe query. But the safety restrictions are ugly! Instead, we will present a different kind of query language, based on algebraic operations rather than first-order logic. And we will show that for any domain independent first-order query there exists an equivalent expression in this algebra. In fact, SQL is inspired by this algebra!

8 Relational Algebra

The relational algebra is a *many-sorted* algebra, where the sorts are the natural numbers. The idea is that the elements of sort n are finite n-ary relations. Recall the domain $\mathbb{D}$. The *carrier* of sort n of the algebra is $REL(\mathbb{D}^n)$ (the set of finite n-ary relations on $\mathbb{D}$).

If f is a many-sorted k-ary operation symbol that takes arguments of sorts $n_1, \ldots, n_k$ (in this order) and returns a result of sort n then we write its type as follows: $f : n_1 \times \cdots \times n_k \longrightarrow n_0$, and we simplify this to n for nullary ($k = 0$) operations.

The operations of the algebra, with their types and their interpretation over the relational carriers are the following:

constant singletons $\{c\} \ : \ 1 \ \ (c \in \mathbb{D})$

selection1 $\sigma^n_{ij} \ : \ n \longrightarrow n \quad (1 \leq i < j \leq n)$ interpreted as $\sigma^n_{ij}(R) = \{\mathbf{x} \in R \mid x_i = x_j\}$.

selection2 $\sigma^n_{ic} \ : \ n \longrightarrow n \quad (1 \leq i \leq n, \ c \in \mathbb{D})$ interpreted as $\sigma^n_{ic}(R) = \{\mathbf{x} \in R \mid x_i = c\}$.

projection $\pi^n_{i_1 \ldots i_k} \ : \ n \longrightarrow k \quad (1 \leq i_1, \ldots, i_k \leq n, \text{ not necessarily distinct})$
interpreted as $\pi^n_{i_1 \ldots i_k}(R) = \{x_{i_1}, \ldots, x_{i_k} \mid \mathbf{x} \in R\}$.

cartesian(cross-) product $\times^{mn} \ : \ m \times n \longrightarrow m + n$
interpreted as $\times^{mn}(R, S) = \{x_1, \ldots, x_m, y_1, \ldots, y_n \mid \mathbf{x} \in R \wedge \mathbf{y} \in S\}$.

union $\cup^n \ : \ n \times n \longrightarrow n$ interpreted as $\cup^n(R, S) = \{\mathbf{x} \mid \mathbf{x} \in R \vee \mathbf{x} \in S\}$.

difference $-^n \ : \ n \times n \longrightarrow n$ interpreted as $-^n(R, S) = \{\mathbf{x} \mid \mathbf{x} \in R \wedge \mathbf{x} \notin S\}$.

Relational algebra *expressions* are built, respecting the sorting, from these operation symbols, using the relational schema symbols *as variables*.

Note that an obvious operation, intersection, is missing. Of course, intersection can be defined from union and difference, by De Morgan's laws. Interestingly, we also have the following:

Exercise 8.1 *Show that intersection is definable just from cartesian product, selection, and projection.*

Given a relational schema Σ, a relational algebra *query* is an algebraic expression constructed from the symbols in Σ and the relational algebra operation symbols, for example if R, S are binary, the expression $\pi_{2414}(\sigma_{13}(R \times S)) - (R \times R)$ defines a query that returns a 4-ary relation (we omit the operation's superscripts because they can usually be reconstructed and we use infix notation for the binary operations). Given a database instance $\mathcal{I}$ as input, such a query e returns a relation $e(\mathcal{I})$ as output.

Clearly, each of the operations of the relational algebra maps finite relations to finite relations, even when the domain of the instance is infinite. In fact, using a definition similar to the one given for first-order queries, the algebra queries are all obviously domain-independent.

Exercise 8.2 *What corresponds in the relational algebra to the FO queries whose output tuple is empty? What happens to the projection operation when $k = 0$?*

Next, we show that the relational algebra and the domain-independent first-order queries have the same expressive power.

Theorem 8.1 *There exists an (easily) computable translation that takes any relational algebra query into an equivalent domain-independent first-order query.*

Proof Sketch By induction on the structure of algebraic expressions. For example, R translates to $\{\mathbf{x} \mid R(\mathbf{x})\}$ while $\{c\}$ translates to $\{x \mid x = c\}$. Here is another case, corresponding to selection1. Suppose that e translates to $q \equiv \{\mathbf{x} \mid \varphi\}$, such that for any instance $\mathcal{I}$, $q(\mathcal{I}) = e(\mathcal{I})$. Then $\sigma_{ij}(e)$ translates to $q' \equiv \{\mathbf{x} \mid \varphi \ \wedge \ x_i = x_j\}$. If q is domain-independent then so is q'. Moreover, for any instance $\mathcal{I}$ $q'(\mathcal{I}) = \sigma_{ij}(e)(\mathcal{I})$. (It is instructive to do the cases corresponding to projection and cartesian product yourselves.) $\square$

The converse is slightly more delicate. Because the set of domain-independent FO queries is not decidable, we cannot define a translation just for these queries. Therefore, we will define a translation for *all* FO queries, such that domain-independent FO queries are translated to equivalent algebraic queries. In fact, we will prove a slightly more general result that will also allow us to transfer undecidability and lower bound results from logic over finite models to databases.

Theorem 8.2 *There exists a computable translation that takes any first-order query q over the schema Σ into a relational algebra query e over the schema $\Sigma \cup \{D\}$ where D is a fresh unary relation symbol, such that for any $\Sigma \cup \{D\}$-instance $\mathcal{I}$, we have*

$$e(\mathcal{I}) = q(\mathcal{I}/D)$$

(Abuse of notation: we denote the interpretation of D in $\mathcal{I}$ also by D.)

Proof Sketch We give the translation by induction on the structure of FO formulas.

Queries defined by atoms such as $\{x_1, \ldots, x_n \mid x_i = c\}$ and $\{x_1, \ldots, x_n \mid x_i = x_j\}$ translate to $\sigma_{ij}(D^n)$, respectively $\sigma_{ic}(D^n)$, where $D^n \equiv D \times \cdots \times D$. Another case: if $\{x_1, \ldots, x_n \mid \varphi\}$ translates to e then $\{x_1, \ldots, x_n \mid \neg\varphi\}$ translates to $e^n_{AD} - e$.

Finally, here is the existential quantifier case. If $\{x_1, \ldots, x_m, z, y_1, \ldots, y_n \mid \varphi\}$ translates to e then $\{x_1, \ldots, x_m, y_1, \ldots, y_n \mid \exists z\, \varphi\}$ translates to $\pi_{1\cdots m(m+2)\cdots(m+n+1)}(e)$. (It is instructive to do the cases corresponding to the relational atom, the conjunction, and the universal quantifier cases.) $\square$

Corollary 8.3 *There exists a computable translation that takes any first-order query q into a relational algebra query e over the same schema such that for any instance $\mathcal{I}$, we have*

$$e(\mathcal{I}) = q(\mathcal{I}/adom(\mathcal{I}) \cup adom(q))$$

(which further equals $q(\mathcal{I})$ whenever q is domain-independent).

Proof Indeed, the active domains can be computed in the relational algebra:

Lemma 8.4 *For every schema Σ there is a relational algebra expression e_{InstAD} (using just cartesian product, projection and union) such that for any Σ-instance $\mathcal{I}$, $e_{InstAD}(\mathcal{I})$ is the active domain of $\mathcal{I}$.*

The proof of the lemma is obvious. Now using the translation provided by the theorem, replace in e the symbol D with the relational algebra expression $e_{InstAD} \cup \{c_1\} \cup \cdots \cup \{c_n\}$ where $c_1, \ldots, c_n = adom(q)$. $\square$

It is possible to define a notion of satisfiability for FO queries and also one of equivalence. But the familiar undecidability demons are still present.

Definition 8.1 A relational algebra expression e is *satisfiable* if there exists an instance $\mathcal{I}$ such that $e(\mathcal{I}) \neq \emptyset$.

Corollary 8.5 *Satisfiability of relational algebra queries is undecidable.*

Proof By reduction from the problem of finite satisfiability of FO sentences (whose undecidability follows immediately from Trakhtenbrot's Theorem). Specifically, we will give a computable translation that takes any first-order sentence φ over a vocabulary $\mathcal{V}$ with just relational symbols and constants into a relational algebra query e over the schema $\mathcal{V} \cup \{D\}$ where D is a fresh unary relation symbol, such that φ is finitely satisfiable iff e is satisfiable.

We use the translation in the theorem for the query $q \equiv \{\ () \mid \varphi\ \}$. The result is a 0-ary relational algebra expression over $\mathcal{V} \cup \{D\}$. Suppose φ is true in the finite $\mathcal{V}$-structure $\mathcal{A}$. We put the elements of the universe of $\mathcal{A}$ in one-to-one correspondence with some elements of $\mathbb{D}$ and consider a database instance $\mathcal{I}$ for the schema $\mathcal{V} \cup \{D\}$ where the relation symbols and constants in φ are interpreted according to the correspondence with $\mathcal{A}$ while D is interpreted as the subset of $\mathbb{D}$ that corresponds to the universe of $\mathcal{A}$. Clearly $q(\mathcal{I}/D) = \{\ ()\ \}$ so $e(\mathcal{I}) = \{\ ()\ \}$ and e is satisfiable.

Conversely, suppose $e(\mathcal{I}) \neq \emptyset$ for some $\mathcal{V} \cup \{D\}$-instance $\mathcal{I}$. It follows that $q(\mathcal{I}/D) \neq \emptyset$. That means that φ is true in the finite structure with universe D (actually, the interpretation of D in $\mathcal{I}$) with the interpretation of the symbols in $\mathcal{V}$ restricted to D. $\square$

Exercise 8.3 *Define* equivalence *of relational algebra queries and prove that it is undecidable.*

9 Complexity of FO Queries

How fast can FO (first-order) queries be evaluated? This is essentially the model checking problem that we have studied earlier. However, we need to take a little care because, as we have defined them, database instances are not the same as arbitrary first-order finite models.

Recall that we have assumed that $\mathbb{D}$ is countably infinite. Let us fix an enumeration [8] of its elements, $a_0, a_1, a_2, \ldots$. We then represent (encode) the interpretation of the relational symbols with respect to this enumeration, just as we did in section 6, with the notable exception that there is no representation of the "size of the universe" (since it is infinite).

Although queries compute *functions* from input instances to output relations, it is more common to study the complexity of an associated decision problem.

Definition 9.1 *The* recognition problem *asks if a tuple u is in $q(\mathcal{I})$ where q is a query and $\mathcal{I}$ an instance. As for model checking, consider as inputs $\langle \mathcal{I}, u, q \rangle$ or $\langle \mathcal{I}, u \rangle$ which jointly encode instances, tuples and queries.*

Theorem 9.1 *For any (fixed) query q the language $\{\langle \mathcal{I}, u \rangle \mid u \in q(\mathcal{I})\}$ is in LOGSPACE. Using the same terminology as in model checking, we say that the* data complexity *of FO queries is in LOGSPACE.*

Proof Sketch This is essentially Theorem 6.2 but we must recall that for model checking the universe of the structure is also part of the input while here it is not. However, given the instance, it is possible to cycle in LOGSPACE through the *active domain*, perhaps more than once through some elements. Since q is domain independent, we need only evaluate it over the active domain. As for u there are two possible ways of dealing it. Can you figure them out? □

Note. The LOGSPACE bound is somewhat conservative. Although we do not know how to describe more accurately the *sequential* complexity of FO queries, we do have a better description of its *parallel* complexity. The class AC_0 consists (roughly) of the languages decided by a parallel random access machine (PRAMs) with polynomially many processors in constant time. ("Roughly", because the precise definition requires that the machine be constructed "uniformly" from the input.)

Theorem 9.2 *The parallel data complexity of FO queries is in AC_0.*

No proof.

Theorem 9.3 *The language $\{\langle \mathcal{I}, u, q \rangle \mid u \in q(\mathcal{I})\}$ is PSPACE-complete. (We say that the* combined complexity *of FO queries is PSPACE-complete.)*

No proof but look again at Theorem 6.2 and at the discussion in the proof of Theorem 9.1 above.

[8] The result of our queries does not depend on the choice of the enumeration; we say that the queries are *generic*. Genericity, regardless of the query language, can be defined precisely (in essence it means invariance under isomorphisms).

Exercise 9.1 *Does the reduction from QBF in the proof of Theorem 6.2 produce domain-independent queries? If not, fix it so it does.*

10 Conjunctive Queries

Conjunctive queries are first-order queries of a particular form: $\{\mathbf{x} \mid \exists \mathbf{y}\, \varphi\}$ where φ is a conjunction of atoms. Example: $\{(x,z) \mid \exists y\, R(x,y) \wedge z = y \wedge S(y,x)\}$. This query is clearly domain-independent, but the following are not: $\{(x,y) \mid R(x)\}$, $\{(x,y) \mid \exists z\, R(x) \wedge y = z\}$ Therefore, we need some additional restrictions to make sure we get only domain-independent queries.

Definition 10.1 A *tableau* T is a set of atoms (relational or equality) that is *range-restricted*, that is, for any variable $x \in \mathit{var}(T)$ either $T \vdash x = c$ for some constant c or $T \vdash x = x'$ for some variable x' that occurs in a relational atom in T.

Here $T \vdash e = e'$ means that $e = e'$ can be derived from the formulas in T via reasoning in FO logic. By completeness, this is the same as $T \models e = e'$. It is clear that the relational atoms in T do not matter and that this kind of consequence is easily decidable, hence so is $T \vdash e = e'$.

Definition 10.2 A *conjunctive query* is a given by a pair $\langle\, \mathbf{u}, T \,\rangle$ where T is a tableau and $\mathbf{u}$ is a tuple of variables or constants (the "output" tuple) [9] such that $\mathit{var}(\mathbf{u}) \subseteq \mathit{var}(T)$. The corresponding FO query is $\{\mathbf{u} \mid \exists \mathbf{y}\, T\}$ where $\mathbf{y} = \mathit{var}(T) \setminus \mathit{var}(\mathbf{u})$ and where we "read" T as the conjunction of its elements.

The range-restriction condition on the tableau part ensures domain-independence (see exercise below).

Here are two examples written in a Prolog-like, or "rule-based", formalism:

$$\begin{array}{lll} \mathit{ans}(x,y) & :- & R(x,z),\ x = c,\ S(x,y,z) \\ \mathit{ans}(c,y) & :- & R(c,z),\ S(c,y,z) \end{array}$$

In the spirit of rule-based/logic programming, the output tuple of conjunctive queries is sometimes called the "head" of the query and the tableau part the "body" of the query.

Notice that the two queries given above are equivalent. It is natural to ask if we can always get rid of equality atoms in a conjunctive query. It turns out that this is the case iff the query is "satisfiable" (see below). Note however that the transformation may introduce constants in the head (as above) or may equate some of the variables in the head, eg.,

$$\begin{array}{lll} \mathit{ans}(x,y) & :- & R(x),\ x = y \\ \mathit{ans}(x,x) & :- & R(x) \end{array}$$

For conjunctive queries without equality atoms we *cannot* assume without loss of generality that the output tuple consists of distinct variables.

[9] Without loss of generality we can assume that the output tuple consists of distinct variables.

We have stated that conjunctive queries are particular cases of FO queries and this defined their semantics. However, it is worthwhile observing that $\mathcal{I}, v \models \exists \mathbf{y} T$ iff $\mathcal{I}, \beta \models T$ for some extension β of v. More precisely:

Definition 10.3 If T is a tableau and $\mathcal{I}$ and instance, a *valuation* for T in $\mathcal{I}$ is a function $\beta : var(T) \longrightarrow \mathbb{D}$. We extend β to map any constant to itself. Moreover, we say that that the valuation β *satisfies* T if

- for any atom $R(\mathbf{e}) \in T$ the relation $R^{\mathcal{I}}$ contains $\beta(\mathbf{e})$, and
- for any atom $e = e' \in T'$ we have $\beta(e) = \beta(e')$.

Proposition 10.1 *For any conjunctive query $q = \langle \mathbf{u}, T \rangle$ and any instance $\mathcal{I}$*

$$q(\mathcal{I}) = \{\beta(\mathbf{u}) \mid \textit{valuation } \beta \textit{ satisfies } T \textit{ in } \mathcal{I}\}.$$

Here is a quick application of this proposition.

Exercise 10.1 *Prove that all conjunctive queries are domain-independent.*

Another application is to characterize and decide satisfiability for conjunctive queries.

Proposition 10.2 *A conjunctive query $q = \langle \mathbf{u}, T \rangle$ is unsatisfiable iff $T \vdash c_1 = c_2$ for two distinct constants c_1, c_2. In particular, satisfiability is easily decidable.*

Proof By proposition 10.1 q is satisfiable iff there exists an instance and a valuation that satisfy T. If $T \vdash c_1 = c_2$ then any valuation must equate c_1 and c_2 which is impossible if they are distinct. This gives us one direction of the proposition. For the converse, suppose that $c_1 \equiv c_2$ whenever $T \vdash c_1 = c_2$. Then, for any $x \in var(T)$ there is at most one constant c such that $T \vdash x = c$. Let γ map every variable in $x \in var(T)$ to such a unique c if it exists. If it doesn't, let γ map x to a fixed constant c_0 that we choose apriori. Extend γ to map any constant to itself. Build an instance $\mathcal{I}_0$ as follows:

$$R^{\mathcal{I}_0} \stackrel{\text{def}}{=} \{\gamma(\mathbf{e}) \mid R(\mathbf{e}) \in T\}$$

We claim that γ is a valuation that satisfies T. The relational atoms are satisfied by construction. If $x = c \in T$ then $\gamma(x) = c = \gamma(c)$. If $x_1 = x_2 \in T$ then (1) either $T \vdash x_1 = c$ for some c in which case we also have $T \vdash x_2 = c$ and therefore $\gamma(x_1) = c = \gamma(x_2)$, (2) or $\gamma(x_1) = c_0 = \gamma(x_2)$. $\square$

Note that proposition 10.2 implies that satisfiability depends only on the tableau part of the query. We can therefore talk about (un)satisfiable tableaux.

Exercise 10.2 *Prove that any equality-free conjunctive query is satisfiable.*

Exercise 10.3 *Prove that for any satisfiable conjunctive query there is an equality-free conjunctive query equivalent to it.*

Proposition 10.1 also implies that conjunctive queries are in NP, which is (probably!) better than general FO queries which are PSPACE-complete (all this is for combined complexity):

Theorem 10.3 *The combined complexity of the recognition problem for conjunctive queries is NP-complete.*

Proof Since the queries are domain-independent, it is sufficient for the valuations to take values in $adom(\mathcal{I}) \cup adom(q)$ instead of all of $\mathbb{D}$. Note that the size of such a valuation is polynomial. Then, to test if $\mathbf{a} \in q(\mathcal{I})$ where $q = \langle\, \mathbf{u}, T \,\rangle$, we can guess such a poly-size finite valuation β and check in polynomial time that it satisfies the relational and equality atoms of T and that $\mathbf{a} = \beta(\mathbf{u})$. Hence the problem is in NP.

To prove NP-hardness, we reduce CLIQUE to our problem. Given a graph G and a number $n > 1$, consider the relational schema with just one binary relation symbol E and construct the instance $\mathcal{I}_G$ that corresponds to the set of edges (pairs of vertices) of the graph G. Then consider the query $\langle\, (), T_{clique} \,\rangle$ where () is the empty (0-ary) tuple, $var(T_{clique}) = \{x_1, \ldots, x_n\}$ and

$$T_{clique} = \bigwedge_{1 \le i \ne j \le n} E(x_i, x_j)$$

Clearly G has a clique of size n iff $() \in \langle\, (), T_{clique} \,\rangle(\mathcal{I}_G)$. $\square$

Exercise 10.4 *The reduction above from CLIQUE is specifically for combined complexity. Prove, with a reduction from 3-colorability, that the expression complexity of the recognition problem is also NP-hard. (It is of course in NP, same proof as above.)*

The conjunctive queries correspond to a specific fragment of the relational algebra, namely the fragment that uses only the selection, projection, and cartesian product operations. We call this fragment the *SPC algebra*.

Theorem 10.4

1. *There is an effective translation that takes every conjunctive query into an equivalent SPC algebra expression.*
2. *There is an effective translation that takes every SPC algebra expression into an equivalent conjunctive query.*

Proof Sketch For example, the query $ans(c, y) :- R(c, z), S(c, y, z)$ is translated to the expression $\pi_{14}(\sigma_{1c}(\sigma_{3c}(\sigma_{25}(R \times S))))$. This suggests the idea behind the translation in part (1) of the theorem: start with the cartesian product of the occurrences of the relations in the body, continue with selections determined by the equality atoms, by the use of the same variable in several relational atoms and by the occurrence of constants in the relational atoms, and end with a projection corresponding to the positions of the variables that appear in the output. For part (2) the translation is defined by induction on SPC algebraic expression and I am going to skip it. Notice however that conjunctive queries always translate into SPC expressions of a special form: a cartesian product followed by

several selections followed by a projection. This is a *normal form* for SPC algebra expressions. By composing the two translations in this theorem we can translate any SPC expression into a normal form. □

Via the translation to the SPC algebra we see that conjunctive queries correspond closely to certain SQL programs. For example, the query $ans(x, y) :- R(x, z), x = c, S(x, y, z)$ corresponds to

```
select r.1, s.2
from   R r, S s
where  r.1=c and s.1=r.1 and r.3=s.2
```

Such SQL programs, in which the "where" clause is a conjunction of equalities arise often in practice. So, although restricted, conjunctive queries are important. But the best evidence for their importance comes from the fact that conjunctive query equivalence is decidable. Indeed, recall that we had an exercise that stated equivalence of relational algebra expressions (therefore FO queries) is undecidable. The decidability of conjunctive query equivalence plays an important role in query *optimization*.

Definition 10.4 A query q is *contained* in another query q', written $q \sqsubseteq q'$, if for any instance $\mathcal{I}$ we have $q(\mathcal{I}) \subseteq q'(\mathcal{I})$. Moreover, q and q' are *equivalent*, written $q \equiv q'$ if both $q \sqsubseteq q'$ and $q' \sqsubseteq q$.

Exercise 10.5 *Prove that for conjunctive queries containment is reducible to equivalence. Is this the case just for conjunctive queries?*

In showing that containment of conjunctive queries is decidable we use the following concept:

Definition 10.5 Given two conjunctive queries $q = \langle \mathbf{u}, T \rangle$ and $q' = \langle \mathbf{u}', T' \rangle$ a *homomorphism* $h : q' \longrightarrow q$ is a mapping $h : var(T') \longrightarrow var(T) \cup \mathbb{D}$, extended to map any constant to itself, and such that

- for any atom $R(\mathbf{e}') \in T'$ we have $T \vdash R(h(\mathbf{e}'))$,
- for any atom $e'_1 = e'_2 \in T'$ we have $T \vdash h(e'_1) = h(e'_2)$, and
- $T \vdash h(\mathbf{u}') = \mathbf{u}$

The following is immediate:

Lemma 10.5

1. *The identity mapping $q \longrightarrow q$ is a homomorphism.*
2. *Homomorphisms compose.*

There are obvious similarities between homomorphisms and valuations. This is not accidental because there is a tight connection between tableaux and database instances. In fact, this connection is the key to the "good behavior" of conjunctive queries.

To every **satisfiable tableau** T we associate a database instance $Inst(T)$ as follows. T (or, more precisely, the set of equality atoms in T) determines an equivalence relation on the set of variables and constants occurring in T, $var(T) \cup adom(T)$. Specifically, e is equivalent to e' iff $T \vdash e = e'$. Let us denote by $\hat{e}$ the equivalence class of the variable or constant e. Because T is satisfiable, such an equivalence class can contain at most one constant. We are going to define $Inst(T)$ by interpreting the relation symbols as sets of tuples of such equivalence classes [10]. Now for any relation symbol R we define

$$\hat{\mathbf{e}} \in R^{Inst(T)} \quad \text{iff} \quad T \vdash R(\mathbf{e})$$

If $\hat{\mathbf{e}} = \hat{\mathbf{e}}'$ then $T \vdash R(\mathbf{e})$ iff $T \vdash R(\mathbf{e}')$ hence this definition does not depend on the choice of representatives from the equivalence classes.

Conversely, to every database instance $\mathcal{I}$ we associate a tableau $Tab(\mathcal{I})$ as follows. The variables of the tableau are the elements of $adom(\mathcal{I})$ and the tableau consists of all the relational atoms $R(\mathbf{a})$ such that $R^{\mathcal{I}}(\mathbf{a})$ holds in $\mathcal{I}$. This tableau is as big as the database itself! Note that there are no equality atoms.

Lemma 10.6

1. *Let $q = \langle\, \mathbf{u}, T \,\rangle$ and q' be two conjunctive queries. Then, there is a homomorphism from q' to q iff $\hat{\mathbf{u}} \in q'(Inst(T))$.*
2. *Let q be a conjunctive query, $\mathcal{I}$ an instance and $\mathbf{a} \in adom(\mathcal{I})$. Then $\mathbf{a} \in q(\mathcal{I})$ iff there is a homomorphism from q to $\langle\, \mathbf{a}, Tab(\mathcal{I}) \,\rangle$.*

Proof Sketch For (part 1) let $q' = \langle\, \mathbf{u}', T' \,\rangle$.

First assume $h : q' \longrightarrow q$. Let the valuation $\hat{h} : var(T') \longrightarrow adom(Inst(T))$ be defined by $\hat{h}(x') = \widehat{h(x')}$. Using the properties of the homomorphism h, it is straightforward to check that $\hat{h}$ satisfies the tableau T'. It follows that $\hat{h}(\mathbf{u}') \in q'(Inst(T))$. But $\hat{h}(\mathbf{u}') = \widehat{h(\mathbf{u}')} = \hat{\mathbf{u}}$.

Conversely, assume $\hat{\mathbf{u}} \in q'(Inst(T))$. Then, there exists a valuation $\beta : var(T') \longrightarrow adom(Inst(T))$ that satisfies T' and such that $\beta(\mathbf{u}') = \hat{\mathbf{u}}$. Define $h : var(T') \longrightarrow var(T) \cup \mathbb{D}$ by choosing $h(x')$ to an element of the equivalence class $\beta(x')$. Again, it is straightforward to check that h is a homomorphism.

(Part 2) follows from (part 1) once we observe that since $Tab(\mathcal{I})$ has no equality atoms there is an isomorphism between $Inst(Tab(\mathcal{I}))$ and $\mathcal{I}$ such that $\hat{\mathbf{a}}$ corresponds to $\mathbf{a}$. $\square$.

Theorem 10.7 *$q \sqsubseteq q'$ iff there is a homomorphism from q' to q.*

Proof Let $q = \langle\, \mathbf{u}, T \,\rangle$ and $q' = \langle\, \mathbf{u}', T' \,\rangle$.

[10] There is a difficulty here, but it is easily overcome. We have said that our database instances are built over $\mathbb{D}$ and these equivalence classes are not elements of $\mathbb{D}$. Well, we can identify the equivalence classes that contain a constant with that unique constant. We can also identify the equivalence classes that contain only variables with some elements of $\mathbb{D}$, chosen outside of $adom(T)$ (this is why we took $\mathbb{D}$ to be an infinite set).

First assume $q \sqsubseteq q'$. Since the identity is a homomorphism, we have by lemma 10.6 (part 1), $\hat{\mathbf{u}} \in q(\mathit{Inst}(T))$. Hence $\hat{\mathbf{u}} \in q'(\mathit{Inst}(T))$ and again by lemma 10.6(part 1) there is a homomorphism from q' to q.

Conversely, assume that $h : q' \longrightarrow q$ is a homomorphism. We wish to show that for any instance and any $\mathbf{a} \in \mathit{adom}(\mathcal{I})$ (this is sufficient because conjunctive queries are domain-independent) we have $\mathbf{a} \in q(\mathcal{I}) \Rightarrow \mathbf{a} \in q'(\mathcal{I})$.

By lemma 10.6 (part 2) if $\mathbf{a} \in q(\mathcal{I})$ then there is a homomorphism $g : q \longrightarrow \langle\, \mathbf{a}, \mathit{Tab}(\mathcal{I}) \,\rangle$. But homomorphisms compose so $g \circ h : q' \longrightarrow \langle\, \mathbf{a}, \mathit{Tab}(\mathcal{I}) \,\rangle$ is also a homomorphism. Again by lemma 10.6 (part 2) we have $\mathbf{a} \in q'(\mathcal{I})$. $\square$.

Since there are only finitely many mappings between variables, containment is decidable. In fact:

Corollary 10.8 *Testing $q \sqsubseteq q'$ is NP-complete.*

Proof By the theorem, this is equivalent to guessing a mapping (clearly of poly-size) from $\mathit{var}(q')$ to $\mathit{var}(q)$ and then checking (clearly in poly-time) that it is a homomorphism. Hence the problem is in NP.

For NP-hardness, we reduce the recognition problem for conjunctive queries to a containment. Indeed, by lemma 10.6 (part 2) $\mathbf{a} \in q(\mathcal{I})$ iff there is a homomorphism from q to $\langle\, \mathbf{a}, \mathit{Tab}(\mathcal{I}) \,\rangle$. By the theorem, this in turn is equivalent to the containment $\langle\, \mathbf{a}, \mathit{Tab}(\mathcal{I}) \,\rangle \sqsubseteq q$. $\square$.

11 Conjunctive Query Minimization

In this section we restrict ourselves to conjunctive queries whose tableau consists only of relational atoms. The treatment can be generalized to deal with equality atoms but the complications might obscure the beauty of the results.

The problem of *query minimization* is the following: given q find an equivalent query q_m that is "minimal". Of course we have to define what "minimal" means, but the intention is that q_m executes as fast as possible. Without going into the subtleties of how to measure/estimate running time for queries, we shall accept the fairly obvious statement that the presence of additional atoms slows down a query. This will give us a partial ordering on queries and then we will look for queries that are minimal with respect to this partial ordering. Formally:

Definition 11.1 *Given a conjunctive query $q = \langle\, \mathbf{u}, T \,\rangle$ a* subquery *of q is a conjunctive query of the form $\langle\, \mathbf{u}, S \,\rangle$ where $S \subseteq T$.*

If q_s is a subquery of q then there is an obvious homomorphism ($x \longmapsto x$) from q_s to q, hence $q \sqsubseteq q_s$. We will be interested in the cases when $q_s \sqsubseteq q$ also holds, hence q_s is actually equivalent to q. It is easy to check that "q_1 is a subquery of q_2 and $q_1 \equiv q_2$" is a partial ordering on queries. The queries that are minimal in this partial ordering shall be called "locally minimal" because none of their atoms can be removed while preserving equivalence.

Definition 11.2 *The query q is* locally minimal *if no strict subquery of q is equivalent to q. (Equivalently, the only subquery of q equivalent to q is q itself.)*

In a first approximation, we could say that to minimize a query q we should try to find locally minimal queries equivalent to q. Obviously, such queries exist because in the partial order we have just defined all descending chains are finite. But "how many" such queries are there? It is trivial that they form an r.e. set: just enumerate all queries up to isomorphism and test each one for local minimality and equivalence to q. But what we wish for is the following situation: there are only finitely many (up to renaming the variables, of course) locally minimal queries equivalent to a given query q and there is an algorithm for computing them. Indeed, we shall see that this is the case, and, as a nice bonus, it will turn out that there is, in fact, only *one* such query!

Consider a homomorphism $h : q' \longrightarrow q$ where $q = \langle\, \mathbf{u}, T \,\rangle$ and $q' = \langle\, \mathbf{u}', T' \,\rangle$. From the definition of homomorphism, the underlying mapping between variables $h : var(T') \longrightarrow var(T)$ also induces a mapping $T' \longrightarrow T$, where $R(\mathbf{x}) \longmapsto R(h(\mathbf{x}))$. With a slight abuse of notation, we shall denote this mapping between atoms also by h.

Lemma 11.1 *Let h be a homomorphism.*

1. *If h is injective on variables then h is injective on atoms.*
2. *If h is surjective on atoms then h is surjective on variables.*

The proof is omitted (but note that (2) makes use of the range-restriction property). Both converses are false.

Exercise 11.1 *Give a homomorphism that is a counterexample to both the converse of (1) and of (2).*

Definition 11.3 *An* isomorphism *is a homomorphism that is a bijection on both variables and atoms. (In view of the lemma above injectivity on variables + surjectivity on atoms suffice.) The notion of isomorphic queries captures our intuition of queries that are the same up to renaming of the variables.*

Lemma 11.2 *A query q is locally minimal iff any homomorphism $q \longrightarrow q$ is an isomorphism.*

Proof First, let $q = \langle\, \mathbf{u}, T \,\rangle$ and suppose any homomorphism $q \longrightarrow q$ is an isomorphism. Let $q_s = \langle\, \mathbf{u}, S \,\rangle$ be a subquery of q such that $q \equiv q_s$. Since $q_s \sqsubseteq q$ there must exist a homomorphism $h : q \longrightarrow q_s$. But h can also be seen as a homomorphism $q \longrightarrow q$ and therefore it must be an isomorphism. Then, $T = h(T) \subseteq S \subseteq T$ hence $S = T$ and q_s is the same as q. Thus, we have shown that q is locally minimal.

Conversely, suppose that $q = \langle\, \mathbf{u}, T \,\rangle$ is locally minimal and let $h : q \longrightarrow q$ be a homomorphism. The pair $\langle\, \mathbf{u}, h(T) \,\rangle$ satisfies the range-restriction condition ($\mathbf{u} = h(\mathbf{u}) \subseteq h(T)$) and therefore $q' = \langle\, \mathbf{u}, h(T) \,\rangle$ is a conjunctive query, in fact a subquery of q. Because of h we have $q_s \sqsubseteq q$ and

hence $q_s \equiv q$. Since q is locally minimal q' must be the same as q hence $h(T) = T$. So h is a surjection $T \longrightarrow T$. It follows that it is also a surjection $var(T) \longrightarrow var(T)$. But both T and $var(T)$ are finite sets hence h is also injective, on both variables and atoms. It follows that h is an isomorphism. □

Theorem 11.3 *Let q be a query. Any locally minimal query equivalent to q is isomorphic to some subquery of q.*

Proof Let $q' = \langle \mathbf{u}', T' \rangle$ be locally minimal and equivalent to $q = \langle \mathbf{u}, T \rangle$. We have a pair of homomorphisms $h : q' \longrightarrow q$ and $g : q \longrightarrow q'$. It follows from lemma 11.2 that $g \circ h : q' \longrightarrow q'$ is an isomorphism. Since $g \circ h$ is injective, h must be injective too.

Consider $\langle \mathbf{u}, h(T') \rangle$, a subquery of q (range-restriction is checked as before). This query is isomorphic to q' via $h : q' \longrightarrow \langle \mathbf{u}, h(T') \rangle$ which is injective and also surjective on atoms hence on variables. □

Therefore, minimization can be handled algorithmically: compute all the subqueries of the given query q and retain the locally minimal ones that are equivalent to q. But we can do better. The same theorem implies the following:

Corollary 11.4 *If q, q' are both locally minimal and equivalent then they are isomorphic.*

Proof By the theorem, q' is isomorphic to some subquery q_s of q. By the local minimality of q, q_s must in fact be q. □

First, this corollary clarifies the role of the alternative notion of "global" minimality. A query is *globally minimal* if it has the minimum number of atoms among all queries equivalent to it. Global minimality clearly implies local minimality and for any query there always exists a globally minimal one equivalent to it. Using the corollary, any locally minimal query is isomorphic to a globally minimal query equivalent to it hence it is globally minimal itself. We see that this alternative notion of minimality coincides with the one we have used.

Second, it follows from this corollary and the theorem that for any query q there is (up to isomorphism) exactly *one* locally minimal query equivalent to q which is moreover a subquery of q. This is called the *core* of q and, as we saw, minimization turns out to be the same as computing the core.

Lemma 11.2 can be used to design an "incremental" algorithm for computing the core:

input: $q = \langle \mathbf{u}, T \rangle$.
  $q_s := \langle \mathbf{u}, T \rangle$
  <u>while</u> there exists some homomorphism $h : q_s \longrightarrow q_s$ that is not an isomorphism
  <u>do</u> $q_s := \langle \mathbf{u}, h(T) \rangle$
output: q_s

At each step, $h(T)$ is a proper subset of T otherwise h is surjective on atoms and hence an isomorphism (by the same argument as in the proof of lemma 11.2). Hence the algorithm always terminates.

Clealry $q \equiv q_s$ is an invariant of the loop in the algorithm. Therefore the algorithm returns a subquery q_s of q that is equivalent to q and such that any homomorphism $h : q_s \longrightarrow q_s$ is an isomorphism. By lemma 11.2 the returned query q_s is also locally minimal and hence it is the core of q.

Exercise 11.2 *Minimize the following query:*

$$ans(x, y, z) :- R(x, y_2, z_2), R(x_1, y, z_1), R(x_2, y_2, z), R(x, y_1, z_1), R(x_2, y_1, z)$$

12 Dependencies

Dependencies are first-order sentences of a particular form (see below). This form is the result of successive generalizations of various definitions of *integrity constraints*, i.e., assertions that database instances are expected to satisfy in order to agree with the semantic intuition of the db designers.

Definition 12.1 *A* conjunctive containment dependency *(*ccd*) is a sentence of the form*

$$(d) \qquad \forall \mathbf{x}(B \to \exists \mathbf{y}\, C)$$

where B and C are tableaux, which are "read" as the conjunction of their elements, and where $\mathbf{x} \subseteq var(B)$ and $\mathbf{y} \subseteq var(C)$. Because (d) is a sentence, we must also have $var(B) \subseteq \mathbf{x}$ and $var(C) \subseteq \mathbf{x} \cup \mathbf{y}$.

The name "ccd" is justified by the following:

Proposition 12.1 *Let $q = (\mathbf{u}, T)$ and $q' = (\mathbf{u}', T')$ be two conjunctive queries such that the tuples $\mathbf{u}$ and $\mathbf{u}'$ have the same width. Let $\{\mathbf{u} \mid \exists \mathbf{z}\, T\}$ (where $\mathbf{z} = var(T) \setminus \mathbf{u}$) and $\{\mathbf{u}' \mid \exists \mathbf{z}'\, T'\}$ (where $\mathbf{z}' = var(T') \setminus \mathbf{u}'$) be the same queries expressed as FO queries. Then*

$$cont(q, q') \stackrel{\text{def}}{=} \forall \mathbf{u} \forall \mathbf{z}(T \to \exists \mathbf{u}' \exists \mathbf{z}'\, T' \wedge \mathbf{u} = \mathbf{u}')$$

is a ccd such that for any instance $\mathcal{I}$ we have

$$q(\mathcal{I}) \subseteq q'(\mathcal{I}) \quad \text{iff} \quad \mathcal{I} \models cont(q, q').$$

It follows that $q \sqsubseteq q'$ iff $\models cont(q, q')$.

Proof Omitted.

Proposition 12.2 *Let $d \stackrel{\text{def}}{=} \forall \mathbf{x}(B \to \exists \mathbf{y}\, C)$ be a ccd and define*

$$front(d) \stackrel{\text{def}}{=} (\mathbf{x}, B) \qquad \text{and} \qquad back(d) \stackrel{\text{def}}{=} (\mathbf{x}, B \cup C)$$

Then, $front(d)$ and $back(d)$ are conjunctive queries (as FO queries they would be written $\{\mathbf{x} \mid B\}$ and $\{\mathbf{x} \mid \exists \mathbf{y}\, B \wedge C\}$) such that

$$\mathcal{I} \models d \quad \text{iff} \quad front(d)(\mathcal{I}) \subseteq back(d)(\mathcal{I}).$$

It follows that d is true in all instances iff the containment $front(d) \sqsubseteq back(d)$ holds. (By the way, note that we have $back(d) \sqsubseteq front(d)$ because B is a subset of $B \cup C$ which gives a trivial homomorphism.)

Proof Omitted.

Corollary 12.3 *Testing whether a ccd holds in* all *instances is decidable, and in fact NP-complete.*

Exercise 12.1 *Show that the problem of testing containment of two conjunctive queries remains NP-complete even when the first query is such that all its variables appear in the output tuple (no existentially quantified variables in the corresponding FO formulation).*

We are, however, interested in a more general, and harder question: given a set of dependencies D and two queries q, q' is it decidable whether $D \models q \sqsubseteq q'$? And further, we are interested in algorithms for minimizing conjunctive queries in the presence of dependencies. The following examples illustrate this.

Example 12.1 Consider the query

$$ans(y, z) :- R(x, y, z'),\ R(x, y', z)$$

As is, it cannot be minimized. But if we assume the dependency (a key)

$$\forall x, y_1, z_1, y_2, z_2 (R(x, y_1, z_1) \wedge R(x, y_2, z_2) \wedge \rightarrow y_1 = y_2)$$

then in all instances in which this dependency holds the query is equivalent to

$$ans(y, z) :- R(x, y, z'),\ R(x, y, z)$$

which can be minimized to a join-less projection:

$$ans(y, z) :- R(x, y, z)$$

Example 12.2 Consider the query

$$ans(x, y) :- R(x, y),\ S(y, z)$$

As is, it cannot be minimized. But if we assume the dependency (like a foreign key)

$$\forall x, y (R(x, y) \rightarrow \exists z\, S(y, z))$$

then in all instances in which this dependency holds the query is equivalent to

$$ans(x, y) :- R(x, y)$$

Note In the rest of this section "query" means conjunctive query and "dependency" means conjunctive containment dependency.

Toward a decision procedure for conjunctive query equivalence under ccd's we develop a technique called the *"chase"*.

In its simplest form, the chase is easy to understand. Consider a query (in FO form) q and a dependency d as follows:

$$q \stackrel{\text{def}}{=} \{\mathbf{x} \mid \exists \mathbf{y}\, B\} \qquad\qquad d \stackrel{\text{def}}{=} \forall \mathbf{x} \forall \mathbf{y} (B \rightarrow \exists \mathbf{z}\, C).$$

When a query and a dependency are syntactically related in this manner a *chase step* from q with d, written $q \xrightarrow{d} q'$, produces the query

$$q' \stackrel{\text{def}}{=} \{\mathbf{x} \mid \exists \mathbf{y} \exists \mathbf{z}\, B \wedge C\}$$

Notice that we have $q' \sqsubseteq q$ and if $\mathcal{I} \models d$ then $q(\mathcal{I}) \subseteq q'(\mathcal{I})$, hence $d \models q \equiv q'$.

An even simpler example of chase step is $\mathit{front}(d) \xrightarrow{d} \mathit{back}(d)$ for any dependency d.

To define the chase step for arbitrary queries and dependencies we need to talk about *homomorphisms of tableaux*. This is the same as what was defined earlier as homomorphism of queries, except that we do not need the third condition (that relates the output variables).

Definition 12.2 *Consider the dependency* $d \stackrel{\text{def}}{=} \forall \mathbf{x}(B \rightarrow \exists \mathbf{y}\, C)$ *and a tableau* T. *We say that the* chase *with* d *is* applicable *to* T *if there exists a homomorphism* $h : B \longrightarrow T$ *that cannot be extended to* $B \cup C$, *i.e., there is* no *homomorphism* $h' : B \cup C \longrightarrow T$ *that is equal to* h *on* $\mathit{var}(B)$. *When the chase is applicable, the* result *of one step of chase of* T *with* d *is the tableau* $T' \stackrel{\text{def}}{=} T \cup C[\mathbf{x} := h(\mathbf{x})]$ *and we write* $T \xrightarrow{d} T'$.

In defining the chase result we assume that $\mathbf{y}$ is disjoint from $\mathit{var}(T)$. If this is not the case we rename the bound variables $\mathbf{y}$. For example, we will certainly need such renaming in a sequence of multiple chase steps with the same dependency.

Recall our earlier construction that associates to every satisfiable tableau T a database instance $\mathit{Inst}(T)$.

Lemma 12.4 *If the chase with* d *is not applicable to a satisfiable tableau* T *then* $\mathit{Inst}(T) \models d$.

Proof We prove the contrapositive. Suppose $\mathit{Inst}(T) \not\models d$.Then, there exists a mapping (a *valuation*) $\beta : \mathbf{x} \longrightarrow \mathit{dom}(\mathit{Inst}(T))$ that satisfies the atoms in B and such that there is no extension $\beta' : \mathbf{x} \cup \mathbf{y} \longrightarrow \mathit{dom}(\mathit{Inst}(T))$ that additionally satisfies the atoms in C. Define $h : \mathit{var}(B) \longrightarrow \mathit{var}(T)$ by choosing $h(x)$ to be some variable in the equivalence class $\beta(x)$. It is straightforward to check that h is a homomorphism of tableaux. This homomorphism cannot be extended to $B \cup C$ because the extension would yield an extension of β that satisfies C. It follows that the chase with d is applicable to T. $\square$.

Definition 12.3 *We define the chase on conjunctive queries using their underlying tableaux: if* $q \stackrel{\text{def}}{=} (\mathbf{x}, T)$ *and* $T \xrightarrow{d} T'$ *then* $q \xrightarrow{d} q'$ *where* $q' \stackrel{\text{def}}{=} (\mathbf{x}, T')$.

Lemma 12.5 *If* $q \xrightarrow{d} q'$ *then* $d \models q \equiv q'$.

Proof Since the tableau of q is a subset of the tableau of q' we have a trivial homomorphism $q \longrightarrow q'$ hence $q' \sqsubseteq q$.

It remains to prove that if $\mathcal{I}$ is an instance such that $\mathcal{I} \models d$ then $q(\mathcal{I}) \subseteq q'(\mathcal{I})$. Let $d = \forall \mathbf{x}(B \rightarrow \exists \mathbf{y}\, C)$, $q = (\mathbf{u}, T)$, and $q' = (\mathbf{u}, T')$. Let $h : B \longrightarrow T$ be the homomorphism used in the chase step hence $T' = T \cup C[\mathbf{x} := h(\mathbf{x})]$. Since the output tuple is the same and $T \subseteq T'$ it suffices to show that any valuation in $\mathcal{I}$ that satisfies T can be extended to a valuation that satisfies T'. Let $\beta : T \longrightarrow \mathcal{I}$ be such a valuation. It follows that $\beta \circ h : B \longrightarrow \mathcal{I}$ satisfies B in $\mathcal{I}$and since $\mathcal{I} \models d$ $\beta \circ h$ can be extended to a valuation γ that satisfies C in $\mathcal{I}$. Now define $\beta' : C[\mathbf{x} := h(\mathbf{x})] \longrightarrow \mathcal{I}$ as follows:

if $z \in \mathbf{y}$ then $\beta'(z) \stackrel{\text{def}}{=} \gamma(z)$ but if $z \in h(\mathbf{x})$ then $\beta'(z) \stackrel{\text{def}}{=} \beta(z)$. Thus β' extends β and what's left is to check that β' satisfies T'. Clearly β' satisfies the atoms that are also in T. Now suppose, (keeping the notation simple) that $R(h(x), y)$ is an atom in $C[\mathbf{x} := h(\mathbf{x})]$ obtained from the atom $R(x, y)$ in C, where $x \in \mathbf{x}, y \in \mathbf{y}$. We have $(\beta'(h(x)), \beta'(y)) = (\beta(h(x)), \gamma(y)) = (\gamma(x)), \gamma(y)) \in R^{\mathcal{I}}$ because γ satisfies C. Similarly for equality atoms. $\square$.

Definition 12.4 *Let q be a conjunctive query and D a set of dependencies. A* terminating chase sequence *of the query q with dependencies from D is a sequence of chase steps*

$$q \stackrel{d_1}{\rightarrow} q_1 \stackrel{d_2}{\rightarrow} q_2 \cdots \stackrel{d_n}{\rightarrow} q_n$$

such that $d_1, \ldots, d_n \in D$ and such that no chase with dependencies from D is applicable to q_n.

Theorem 12.6 *Let q, q' be two conjunctive queries and let D be a set of dependencies such that there exists a terminating chase sequence of q with D. Let q_n be the last chase result in the sequence. Then*

$$D \models q \sqsubseteq q' \quad \textit{iff} \quad q_n \sqsubseteq q$$

Proof One of the implications follows from the fact that $D \models q \equiv q_n$ (see the lemma above).

For the other implication, let T_n be the tableau underlying q_n and let $\mathcal{I}_n \stackrel{\text{def}}{=} \mathit{Inst}(T_n)$. If T_n is unsatisfiable the q_n returns the empty set on all inputs hence it is contained in any query, in particular q. Assume that T_n is satisfiable. By the other lemma above, $\mathcal{I}_n \models D$, hence $\mathcal{I}_n \models q \sqsubseteq q'$.

Now let $q \stackrel{\text{def}}{=} (\mathbf{u}, T)$ and $q' \stackrel{\text{def}}{=} (\mathbf{u}', T')$ Note that $T \subseteq T_n$ because each chase step just adds atoms. So there is a mapping $\sigma : \mathit{var}(T) \longrightarrow \mathit{dom}(\mathcal{I}_n)$ that associates each variable to its equivalence class, $\sigma(x) = \hat{x}$. It follows that $\hat{\mathbf{u}} = \sigma(\mathbf{u})$ is in $q(\mathcal{I}_n)$ and therefore also in $q'(\mathcal{I}_n)$. Hence there must exist a mapping $\beta : \mathit{var}(T') \longrightarrow \mathit{dom}(\mathcal{I}_n)$ such that $\beta(\mathbf{u}') = \hat{\mathbf{u}}$. It is now straightforward to show that we have a homomorphism $q' \longrightarrow q_n$. $\square$.

Hence, we will be interested in terminating chase sequences. There exist sets of dependencies for which no chase sequence terminates. Consider

$$\begin{array}{ll} (d_1) & \forall x, y(R(x, y) \rightarrow \exists z\, S(z, y)) \\ (d_2) & \forall x, y(S(x, y) \rightarrow \exists z\, R(y, z)) \end{array}$$

Up to renaming the fresh variables there is just one chase sequence for the tableau consisting of the atom $R(u_0, u_1)$, and it is infinite:

Chase step with	adds the atom
(d_1)	$S(v_1, u_1)$
(d_2)	$R(u_1, u_2)$
(d_1)	$S(v_2, u_2)$
(d_2)	$R(u_2, u_3)$
$\cdots$	$\cdots$

Definition 12.5 *A dependency is* full *if it does not have any existentially quantified variables.*

Lemma 12.7 *Any chase sequence with full dependencies terminates.*

Proof Let $d = \forall \mathbf{x}(B \to C)$ be a full dependency. To each satisfiable tableau T we associate a pair of natural numbers (m, n) such that

m is the number of equivalence classes on $var(T) \cup adom(T)$ determined by the $T \vdash e = e'$ equivalence relation (recall the construction of $Inst(T)$).

n is the number of k-tuples built from the same equivalence classes but do *not* hold in $Inst(T)$, where k ranges of the arities of the relation symbols in the schema.

We order these pairs lexicographically, i.e., $(m_1, n_1) < (m_2, n_2)$ iff $m_1 < m_2$ or $m_1 = m_2$ and $n_1 < n_2$. Clealry, there cannot be infinite strictly descending sequences of pairs in this ordering. Now the result follows from the following claim:

Suppose that $T \xrightarrow{d} T'$ and let (m, n) and (m', n') be the pairs of numbers associated to T and T' respectively. Then $(m, n) > (m', n')$.

Indeed, suppose we have an equality atom that is provable from T' but not from T. Then $m > m'$. Otherwise, T and T' determine the same equalities so $Inst(T)$ and $Inst(T')$ are built on the same set of equivalence classes and $m = m'$. Let $h : B \longrightarrow T$ be the homomorphism used in $T \xrightarrow{d} T'$. By definition of the chase step, h is *not* a homomorphism from $B \cup C$ to T. So there must exist an atom $R(\mathbf{e}) \in C$ such that $T \nvdash R(h(\mathbf{e}))$ while, of course, $R(h(\mathbf{e})) \in C[\mathbf{x} := h(\mathbf{x})] \subseteq T'$. Hence $R(\widehat{h(\mathbf{e})})$ holds in $Inst(T')$ but not in $Inst(T)$ so $n > n'$. $\square$.

Corollary 12.8 *If the dependencies in D are full then testing $D \models q \sqsubseteq q'$ is decidable.*

Even when the dependencies are not full there are more general conditions on sets of dependencies that guarantees termination.

First a bit of terminology: for a dependency $\forall \mathbf{x}(B \to \exists \mathbf{y}\, C)$ we call B the *premise* of the dependency and C the *conclusion* of the dependency.

For simplicity we assume that there are no equality atoms in the dependencies.

Given a finite set of dependencies D build the following *chase-flow* graph from D:

- For each relation symbol of arity k that appears in D and each $1 \leq i \leq k$ build a node labeled (R, i).
- For each dependency and each variable x that occurs in position i in an R-atom in the premise and in position j in an S-atom in the conclusion draw an edge labeled $\forall$ between (R, i) and (S, j).
- For each dependency and each variable x that occurs in position i in an R-atom in the premise and each variable y that occurs *existentially quantified* in position j in an S-atom in the conclusion draw an edge labeled $\exists$ between (R, i) and (S, j).

Definition 12.6 (Deutsch& Popa) *The set D of dependencies is said to be* weakly acyclic *if the associated chase-flow graph does* not *have a cycle in which at least one edge is labeled $\exists$.*

Theorem 12.9 (Deutsch& Popa) *If D is weakly acyclic then all chase sequences with D terminate.*

The proof is omitted.

Note It was shown that the problem of deciding conjunctive query containment under ccds is EXPTIME-complete. However, this is one such problem that can actually be solved in practice, because queries and dependencies are not so big and the algorithm can be heavily "engineered" to run faster. It turns out that this works for queries of up to 20 atoms and for hundreds of dependencies.