Core Finding for Relational Structures

A Major Qualifying Project Report
submitted to the faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements
for the degree of Bachelor of Science

in

Computer Science

by

VISIT PATARANUTAPORN

May 1, 2014

APPROVED:

Professor Daniel Dougherty, project advisor

Abstract

The computation of "cores" of relational structures has a variety of applications. In this project, we revise a core computation algorithm by Pichler and Savenkov, designed for the data exchange context, to work in a more general setting. Contributions to research from this project include the observation that the previous algorithm by Pichler and Savenkov may not work when disjunctions are present in theories, a revised algorithm for the new setting, and an implementation of the algorithm in Haskell. We use "signature testing" as a heuristic to improve the running time of the algorithm.

Contents

Chapter 1

Introduction

1.1 Problem Domain

This project involves core computation of abstract structures, or “models.” Examples of abstract structures include graphs and databases. There are studies about cores of graphs [9] and cores in the data exchange problem [8]; the latter will be touched upon soon below.

1.2 Specific Problem

The focus of this project is on core computation for relational models, specifically those generated by the chase algorithm [1, 10]. These *chase models* are special cases of relational models because they have special properties, such as being homomorphically minimal compared to relational models in general, as will be explained in Section 2.1.

1.3 State of the Art

Core-finding algorithms originated in graph theory [9], but the algorithm used in this paper was invented in the area of data exchange [5, 6, 7, 8, 11], a problem in database theory. Early algorithms were invented by Fagin et al. [5] for special cases of database constraints. An algorithm FindCore by Gottlob and Nash [8] works for a setting where a set of constraints consists of a weakly acyclic set of tuple-generating dependencies (*tgds*) and a set of equality-generating dependencies (*egds*). In their algorithm [8], they used special tgds to simulate egds, thus treating egds indirectly. The algorithm runs in $O(n^c)$, where n is the size of the initial source instance (model) and c is a constant that depends only on the set of constraints. Afterwards, Pichler and Savenkov [11] presented FindCore^E, an improved version of FindCore [8], by treating egds directly, where running time improved by an order of magnitude in practice. Note that both FindCore and FindCore^E run in polynomial time in the instance size (the number of facts).

In this paper, we treat a more general class of constraints (see Section 2.1). However, our algorithm for this setting may not run in worst case polynomial time.

1.4 Problems to be Solved

The goals of this project are to revise the previous core-finding algorithms from the data exchange context to work in a generalized setting, which allows disjunctions in geometric theories (to be defined in Section 2.1); and to implement this algorithm.

1.5 Contributions

This project resulted in some contributions to research in model-finding. First are observations, such that the previous algorithm FindCore^E by Pichler and Savenkov [11], though correct in the data exchange setting, may fail when disjunctions are present in theories. Second, the algorithm FindCore^E is extended to FindCore^D, which allows disjunctions in a theory (hence “D” for disjunction), thus working in a more general setting. Finally, the revised algorithm is implemented in Haskell. The implementation contains “signature testing” as a heuristic to prune search space and thus improve the running time of the algorithm.

1.6 Motivation/Applications

At WPI, there is a research project named Hominy, which is a first-order model-finding algorithm based on the Chase [1, 10]. Since the chase algorithm is also used to solve the data exchange problem [4], the nature of models by Hominy is similar to that of solutions in data exchange. Thus, previous core-finding algorithms can be adapted and incorporated into Hominy, allowing Hominy to produce core models.

1.7 Structure of the Paper

In Chapter 2, mathematical and research background about core computation is covered, followed by previous algorithms in Chapter 3. Then in Chapter 4, a revised algorithm and implementation details are discussed. The implementation will be evaluated in Chapter 5, which includes experiments and results. Finally, the paper concludes in Chapter 6.

Chapter 2

Background

2.1 Model-Finding Problem

2.1.1 First-Order Language

Fix a first-order language $\mathcal{L}$. The language of logic formulae used in this paper consists only of relation (predicate) symbols, but no function symbol or constants. Each relation symbol R has an *arity* $n \geq 0$.

2.1.2 Models, Facts, and Elements

Definition 2.1. Given $\mathcal{L}$, a *model* M for $\mathcal{L}$ consists of

1. a domain set $|M|$ or $dom(M)$,
2. for each relation symbol $R \in \mathcal{L}$ of arity n, a relation

$$R(M) \subseteq |M|^n.$$

When a tuple $\vec{a} = (a_1, a_2, \ldots, a_n) \in R(M)$, we call "$R(\vec{a})$" a *fact* and each $a_i \in \vec{a}$ an *element*. In our models, every element will participate in at least one relation, so we may identify a model with its set of facts.

As such, only relational models are considered here. Furthermore, the notion of facts is extended to contain an *id* [11], which is slightly differently defined below. IDs are used to distinguish facts with the same relation name and tuple and relate facts with the same id to one another, both of which cases may arise in the algorithm FindCore^E [11].

Definition 2.2. Let M be a model. *Fact ids* (*fid*s) are unique identifiers for facts in M, i.e., there is a surjective function f that maps a *fid* to a fact in M. For any facts $A, B \in M$, A and B are the same fact if and only if $fid_A = fid_B$.

2.1.3 Geometric Sequents and Theories

Definition 2.3. Fix a language $\mathcal{L}$. A *geometric sequent* is a first-order formula

$$\sigma : \forall \vec{x}.\Phi(\vec{x}) \to \Psi(\vec{x}), \tag{2.1}$$

where the *premise* $\Phi(\vec{x})$ and the *conclusion* $\Psi(\vec{x})$ are formulae built from conjunctions ($\wedge$), disjunctions ($\vee$), equalities ($=$) between variables, atomic formulae, truth ($\top$), falsehood ($\bot$), and existential quantifiers ($\exists$), and all variables are bounded.

A set of geometric sequents $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_s\}$ is called a *geometric theory*, and if M satisfies each sequent of Σ, we write "$M \models \Sigma$" and we say "M is a model of Σ."

Conversion to Restricted Geometric Sequents

This small section covers useful steps to transform general geometric sequents and formulae into a special form described after this section:

- **Converting Φ_σ and Ψ_σ into disjunctive normal form (DNF)**: A formula in DNF is of the form

$$\bigvee_i \left(\bigwedge_j a_{i,j} \right),$$

where $a_{i,j}$ is either an atom or an equality; i.e., disjuncts of conjuncts. Any formula a of the form

$$\left(\bigvee_i p_i\right) \wedge \left(\bigvee_j q_j\right),$$

where p_i and q_j are formulae, is equivalent to

$$\bigvee_{i,j} (p_i \wedge q_j)$$

by the distributive property of formulae. Thus, disjunctions can always be pulled out from conjuncts, even though the number of atoms may increase drastically. Finally, one can repeat the procedure above to pull out disjunctions to a higher level until both Φ_σ and Ψ_σ are in DNF.

- **Eliminating $\vee$ from Φ_σ**: Let σ be of the form $(\bigvee_i \phi_i) \to \Psi_\sigma$ where each ϕ_i is a (possibly existentially quantified) conjunction of atoms and equalities. Then σ can be replaced by a set of sequents $\Sigma_\sigma = \{\sigma'_i : \phi_i \to \Psi_\sigma\}$. More precisely, reassign $\Sigma := (\Sigma - \{\sigma\}) \cup \Sigma_\sigma$.

- **Eliminating $\top$ and $\bot$**: Let a be any atomic formula. Then any occurrence of $\top$ or $\bot$ that is not alone in either the premise Φ_σ or the conclusion Ψ_σ can be eliminated using the following identities.

$$a \wedge \top = \top \wedge a = a$$

$$a \vee \top = \top \vee a = \top$$

$$a \wedge \bot = \bot \wedge a = \bot$$

$$a \vee \bot = \bot \vee a = a$$

Moreover, if Φ_σ only contains $\bot$ or Ψ_σ only contains $\top$, then for any model M, $M \models \sigma$, and thus σ can be removed from Σ without affecting the set of models of Σ. If $\perp$ is in Ψ_σ, then there will not be any model M such that $M \models \Phi_\sigma$.

- **Moving existential quantifiers to the conjunctional level of Φ_σ or ψ_σ:** Let a be a conjunction of atomic formulae, i.e., of the form

$$a : \bigwedge_i \left(\exists \vec{y}_i P_i\left(\vec{x}, \vec{y}_i\right)\right),$$

where $\vec{x}$ is a list of free variables and each $P_i\left(\vec{x}, \vec{y}_i\right)$ is an atomic formula. Then a is equivalent to the formula

$$\exists \vec{y}\left[\bigwedge_i P_i\left(\vec{x}, \vec{y}_i\right)\right],$$

where $\vec{y}$ consists of all variables in each $\vec{y}_i$ and these variables are renamed in such a way that if $y' = y_1 \in \vec{y}_j$ and $y' = y_2 \in \vec{y}_k$, $j \neq k$, then y_1 and y_2 are distinct in $\vec{y}$. (Equalities may be present in Φ_σ or ψ_σ as additional conjuncts that are "anded" with a, but they should be separated since they will be dealt with later in some step.) Furthermore, if Ψ_σ is of the form

$$\Psi_\sigma : \exists \vec{z}\left[\bigvee_i \psi_i\left(\vec{x}, \vec{z}\right)\right],$$

then its $\exists \vec{z}$ can be moved into the disjunction, i.e.,

$$\bigvee_i \left[\exists \vec{z} \psi_i\left(\vec{x}, \vec{z}\right)\right].$$

- **Eliminating $\exists$ from Φ_σ:** Let σ be of the form

$$\forall \vec{x}\left[\exists \vec{y} \phi\left(\vec{x}, \vec{y}\right) \rightarrow \Psi\left(\vec{x}\right)\right],$$

where $\phi\left(\vec{x}, \vec{y}\right)$ is a conjunction of atoms and equalities without any existential quantification. Then σ is equivalent to a sequent

$$\forall\vec{x}\forall\vec{y}\left[\phi\left(\vec{x},\vec{y}\right)\rightarrow\Psi\left(\vec{x}\right)\right],$$

which can be derived as follows:

$$\forall\vec{x}\left[\exists\vec{y}\phi\left(\vec{x},\vec{y}\right)\rightarrow\Psi\left(\vec{x}\right)\right]$$

$$\forall\vec{x}\left[\left(\neg\exists\vec{y}\phi\left(\vec{x},\vec{y}\right)\right)\vee\Psi\left(\vec{x}\right)\right]$$

$$\forall\vec{x}\left[\left(\forall\vec{y}\left[\neg\phi\left(\vec{x},\vec{y}\right)\right]\right)\vee\Psi\left(\vec{x}\right)\right]$$

$$\forall\vec{x}\forall\vec{y}\left[\neg\phi\left(\vec{x},\vec{y}\right)\vee\Psi\left(\vec{x}\right)\right]$$

$$\forall\vec{x}\forall\vec{y}\left[\phi\left(\vec{x},\vec{y}\right)\rightarrow\Psi\left(\vec{x}\right)\right].$$

- **Eliminating $\exists$ from equalities (in ψ_σ)**: Let e be an equality. If e is of the form

$$e:\exists y.x=y,$$

  where x is a free variable, then by substituting y with x and by removing $\exists y$, e will become a tautology $x=x$ for any free variable x. The form $\exists y.y=x$ is similar.

- **Eliminating equalities from Φ_σ**: Let σ be of the form

$$\sigma:\forall\vec{x}\left[\bigwedge_i\left(u_i=v_i\right)\wedge\phi\left(\vec{x}\right)\rightarrow\Psi\left(\vec{x}\right)\right],$$

  where u_i and v_i are variables in $\vec{x}$ and $\phi\left(\vec{x}\right)$ is a conjunction of atoms. Then, by replacing u_i with v_i within $\vec{x}$, for every i, those equalities will be equivalent to $\top$ and thus can be removed from the premise.

Restricted Geometric Sequents

Without loss of generality, any general geometric sequent in (2.1) above can be transformed into a more restricted form using the steps we described. This form is useful and convenient to use in this paper. In this restricted form, premises Φ and conclusions Ψ in Σ are of the form

$$\Phi(\vec{x}) = \bigwedge_i P_i(\vec{x})$$

$$\Psi(\vec{x}) = \bigvee_i \psi_i(\vec{x}).$$

Moreover, each conjunct $P_i(\vec{x})$ is an atomic formula, and each disjunct ψ is of the form

$$\psi(\vec{x}) = \psi_{eq}(\vec{x}) \wedge \exists \vec{y} \left[\psi_{pred}(\vec{x}, \vec{y})\right],$$

where

$$\psi_{eq}(\vec{x}) = \bigwedge_j x_{a_j} = x_{b_j},$$

$$\psi_{pred}(\vec{x}, \vec{y}) = \bigwedge_j Q_j(\vec{x}, \vec{y}),$$

$x_a = x_b \, (x_a, x_b \in \vec{x})$ is called an *equality*, $\vec{y}$ is possibly empty, $Q_j(\vec{x}, \vec{y})$ is an existentially quantified atomic formula of relation symbol $Q_j \in \mathcal{L}$, and each variable $y \in \vec{y}$, $y \notin \vec{x}$, is bound to the existential quantifier. Also note that in the data-exchange setting [4], the conclusion consists of only one disjunct, and as special (but wide) cases, a sequent with an equality and a sequent only with existentially quantified atomic formulae are called an *egd* and a *tgd*, respectively [2]. Finally, for convenience, the universal quantifier $\forall \vec{x}$ of σ in (2.1) is usually implied and will be omitted.

One special class of geometric theories is called the *data exchange problem.* Details about the problem can be found in the paper by Fagin et al. [4]. This problem is not the focus of this paper but is mentioned here since the core-finding algorithm in this paper is adapted from those [8, 11] used in the context of data exchange.

2.1.4 Homomorphism, Cores, and Sets of Supports

Definition 2.4. Let M be a model for a theory Σ. A *homomorphism* $h : M \to M'$ is a mapping $dom(M) \to dom(M')$ such that for every fact $P(\vec{a}) \in M, P(h(\vec{a})) \in M'$, and for every constant $c \in dom(M)$, $h(c) = c$. An *endomorphism* is a homomorphism $h : M \to M$. A *retraction* is an endomorphism $r : M \to M$ such that $r \circ r = r$. A *retract* of M is the image $r(M)$ of a retraction r of M. Finally, M_C is a *core* if every endomorphism on it is an isomorphism. There may exist multiple cores of M, but they are unique up to isomorphism.

The following theorem is useful in proving the correctness of our algorithm (Section 4.4).

Theorem 2.5. *Let Σ be a geometric theory. If $A \models \Sigma$ and B is a retract of A, then $B \models \Sigma$, i.e., Σ is closed under retractions.*

Proof. (cf. [8]) Let σ be a geometric sequent of the form

$$\sigma : \Phi(\vec{x}) \to \bigvee_i \psi_i(\vec{x}),$$

where

$$\psi_i(\vec{x}) = \psi_{i,eq}(\vec{x}) \wedge \exists \vec{y} \left[\psi_{i,pred}(\vec{x}, \vec{y})\right].$$

Let $A \models \sigma$ and let $r : A \to B$ be a retraction such that $r(A) = B$. If $B \models \Phi\left(\vec{b}\right)$ for some $\vec{b} \in dom(B)$, then also $A \models \Phi\left(\vec{b}\right)$ since r is a retraction. Since $A \models \sigma$, there exists k such that $A \models \psi_k\left(\vec{b}\right)$, i.e., $A \models \psi_{k,eq}\left(\vec{b}\right)$ and there exists $\vec{a} \in dom(A)$ such that $A \models \psi_{k,pred}\left(\vec{b}, \vec{a}\right)$. This implies that $B \models \psi_{k,eq}\left(r\left(\vec{b}\right)\right) \wedge \psi_{k,pred}\left(r\left(\vec{b}\right), r\left(\vec{a}\right)\right)$. Finally, since r is a retraction and $\vec{b} \in dom(B)$, we have $r\left(\vec{b}\right) = \vec{b}$ and $B \models \psi_{k,eq}\left(\vec{b}\right) \wedge \psi_{k,pred}\left(\vec{b}, \vec{c}\right)$ for $\vec{c} = r(\vec{a})$, i.e., $B \models \sigma$ as required. $\square$

The following lemma is an observation about the existence of non-injective endomorphisms and will be used in the revised algorithm (Section 4.3).

Lemma 2.6. *Let M be a model for a geometric theory Σ. Then there exists a non-injective endomorphism h on M if and only if there exists an endomorphism h' on M and elements $u, v \in dom(M), u \neq v$, such that $h'(u) = h'(v) = v$.*

Proof. Let h be an endomorphism where $h(x) = h(y)$ for some $x, y \in dom(M)$. There are two cases to consider. The case for the backward implication is trivial: If such h', u, and v exist, then we can assign $h := h'$, $x := u$, and $y := v$, and therefore, we have the endomorphism h as required. In the other case, assume that there exist an endomorphism h on M and distinct elements x, y such that $h(x) = h(y)$. By Theorem 5 in Gottlob and Nash [8] (Theorem 3.4 on page 23), h can be transformed into a retraction r such that $r(x) = r(y)$. Let $a = r(x)$. Since $x \neq y$, we have $a \neq x$ or $a \neq y$. Without loss of generality, let us assume that $a \neq x$ (if $a = x$, just swap x and y to get $a \neq x$). Then assign $u := x$, $v := a$, and $h' := r$. Since $r(x) = r(a) = a$, we have $h'(u) = h'(v) = v$ with $u \neq v$ as required. □

Definition 2.7. A *set of support* of a theory Σ is a set $S = \{M \mid M \models \Sigma\}$ such that for any model $A \models \Sigma$, there exists a model $M \in S$. A *minimal set of support* S_{min} of Σ is a set of support of Σ such that there is no proper subset that is a set of support of Σ.

Note that in the non-disjunction case, S_{min} contains at most one model called a *universal solution* [4]. However, since that is not the case for this paper, the terminology does not apply here.

2.1.5 Element Rigidity

Definition 2.8. ([5]): Let M be a model. An element $e \in dom(M)$ is *rigid* if for any endomorphism $h : M \to M, h(e) = e$.

Note that in this paper, “element signatures” (defined in Section 4.2) can give a sufficient condition the rigidity of elements.

2.1.6 Blocks, Non-Rigid Blocks, and Block Size

Definition 2.9. Let M be a model and let the *Gaifman graph* $\mathcal{G}(M)$ of M be defined as such a graph (V, E) where vertices in V are elements and an edge (a, b) is in E if there exists a fact $P(\vec{c}) \in M$ such that $a, b \in \vec{c}$. A *block* of M is a connected component of $\mathcal{G}(M)$, thus a set of “tangle” elements. A *non-rigid block* of M is a connected component of the $\mathcal{G}'(M)$ whose vertices are restricted to *non-rigid* elements of M (cf. [11]). Finally, the *block size* of M is the maximal number of elements in any block of M.

2.2 The Chase Algorithm

2.2.1 Chase

The *Chase* is an iterative algorithm invented by Aho et al. [1] and Maier et al. [10]. Not only can the chase solve the data exchange problem [4, 5], but it can also find models for geometric theories. In the latter case, it works as follows.

Given a current model M and a theory Σ, if $M \models \Sigma$, then the chase terminates, and M is a model of Σ. Otherwise, let $\sigma : \Phi(\vec{x}) \to \Psi(\vec{x})$ be a sequent such that $M \not\models \sigma$, i.e., there exists a variable assignment $\vec{a}$ of $\vec{x}$ such that $M \models \Phi_\sigma(\vec{a})$ but $M \not\models \Psi_\sigma(\vec{a})$. Then, the chase (said to *fire the sequent* σ) selects one disjunct (branch) $\psi_\sigma(\vec{a})$ and creates a new model M' from M as follows from (1) to (2):

1. Let $\psi_{\sigma,pred}(\vec{x}, \vec{y}) = \bigwedge_j Q_j(\vec{x}, \vec{y})$. Since $\exists \vec{b}. \bigwedge_j Q_j(\vec{a}, \vec{b})$ is false, therefore:

    (a) New elements $\vec{b}$ all not already in $dom(M)$ will be added to $dom(M')$, i.e., $dom(M') := dom(M) \cup \left\{b \mid b \in \vec{b}\right\}$.

(b) All facts in $\left\{Q_j(\vec{a},\vec{b})\right\}$ will also be added to M', i.e., $M' := M \cup \left\{Q_j(\vec{a},\vec{b})\right\}$.

2. Then after (1), for each element pair (a, b) in $\psi_{\sigma,eq}(\vec{a})$, $a, b \in \vec{a}$ and $a \neq b$, all occurrences of a will be replaced with b (or vice versa) in M', pair after pair in any order.

Finally, models generated by the chase will be referred to as *chase models*, which are the focus of this paper.

There are three possible outcomes for a run of the Chase on geometric theories:

1. the algorithm terminates with a model, which satisfies Σ,

2. it terminates with failure, i.e., no model, and

3. it never terminates (an infinite loop).

Theorem 2.10. *Let Σ be a geometric theory. Then Σ is satisfiable if and only if there is a fair run of the Chase which does not fail. A (possibly infinite) set of models obtained by some execution of the Chase is a set of support of Σ.*

In the literature, there is a useful sufficient condition for the termination of the Chase, *weak acyclity* [3, 4], which is defined as the following (borrowing some terms from the paper by Pichler and Savenkov [11]).

Definition 2.11. ([3, 4]) Fix a language $\mathcal{L}$. Let Σ be a geometric theory. Let a *field* R^i be defined for each position (index) i of the relation symbol $R \in \mathcal{L}$. Let $G^D = (V^D, E^D)$ be the *dependency graph* of Σ where vertices of V^D are all the fields R^i in Σ and the set E^D consists of two types of edges, normal edges E^D_n and "existential" edges E^D_e, i.e., $E^D = E^D_n \cup E^D_e$. E^D_n contains an edge (R^i, S^j) if there exists a sequent σ such that the variable $x \in \vec{x}$ in the field R^i in $\Phi_\sigma(\vec{x})$ is also in the field S^j in $\Psi_\sigma(\vec{x})$. E^D_e contains an edge (R^i, S^j) if there exists a sequent σ such that the variable $x \in \vec{x}$ in the field R^i in $\Phi_\sigma(\vec{x})$ is also in some field in $\Psi_\sigma(\vec{x})$ and the variable y in the field S^j is bound in $\Psi_\sigma(\vec{x})$. If for every cycle $\langle v_1, v_2, \ldots, v_k, v_{k+1} = v_1 \rangle \in G^D$, $(v_i, v_{i+1}) \notin E_e^D$, then Σ is *weakly acyclic*.

As such, this paper only focuses on *geometric* theories that are *weakly acyclic*.

The two following definitions mostly follow Pichler and Savenkov [11].

2.2.2 Siblings, Parents, Origins, and Ancestors

Definition 2.12. Let Σ be a theory and M be the current model, while running the Chase, such that $M \not\models \Sigma$. Let $\sigma : \Phi(\vec{x}) \to \Psi(\vec{x})$ be a sequent in Σ and let $\vec{a}$ be a tuple of elements of M such that $M \models \Phi(\vec{a})$ but $M \not\models \Psi(\vec{a})$. Finally, let $\psi_{\sigma,pred}(\vec{a}, \vec{b}) = \bigwedge_j Q_j(\vec{a}, \vec{b})$ be in a branch chosen by the chase in firing σ. Then, facts in $\left\{Q_j(\vec{a}, \vec{b})\right\}$ are *siblings*, newly created facts to be added to M; all elements in $\vec{b}$ are also new. For each element $e \in \vec{b}$, we define its *origin fact origin*(e) to be $Q_k(\vec{a}, \vec{b})$ for some k such that e participates in $Q_k(\vec{a}, \vec{b})$; for each element $e \in \vec{a}$, *origin*(e) can be traced back to some previous sequent firing of σ' with $\vec{a}', \vec{b}'$ where $e \in \vec{b}'$ participates in $\psi_{\sigma',pred}(\vec{a}', \vec{b}')$. The *parents* set of a fact $P\left(\vec{d}\right)$ is defined as the set $parents\left(P\left(\vec{d}\right)\right) = \left\{origin(e) \mid e \in \vec{d} \wedge origin(e) \neq P\left(\vec{d}\right)\right\}$. Finally, let the parent relation be defined on $facts(M) \times facts(M)$ where a pair (p, q) in the relation if and only if $q \in parents(p)$. Then, the *ancestor* relation on facts is the transitive closure of the parent relation.

The following definition describes an important construct used in FindCore^E [11], which will be described in Chapter 3. Note that this definition is slightly different from theirs, in that our "starting subset" T is always empty while it is not the case for FindCore^E, and that we write M_{xy} for T_{xy}.

Definition 2.13. Given a geometric theory Σ, a model M, and elements $x, y \in M$, then M_{xy} is defined as the smallest set containing facts *origin*(x)

and $origin(y)$ such that if any fact A is in M_{xy}, then $siblings(A) \cup parents(A) \subseteq M_{xy}$ (closure under parents and siblings).

2.2.3 Provenance Information

Definition 2.14. Let Σ be a geometric theory, let M be a model (possibly partial, or $M \nvDash \Sigma$), and let $A \in M$ be a fact from firing the sequent $\sigma_k : \Phi(\vec{x}) \to \Psi(\vec{x})$ with variable substitutions $\vec{x} = \vec{a}$, where this is the sth sequent firing of Σ. *Provenance information* $prov_A$ of a fact A is a triplet (s_A, k_A, b_A) where

1. $s_A = s$ is a sibling id, or *sid*, which is the ordinal number of the sequent firing,

2. $k_A = k$ is a *sequent number*, which is a unique identifier for sequents, and

3. b_A is a *variable binding* or *substitution*, which is a set of pairs $(x_i, a_i), x_i \in \vec{x}, a_i \in \vec{a}$, for each i.

Moreover, for each $B \in M$, $B \in siblings(A)$ if and only if $s_A = s_B$; and for any $C, D \in M$, if $k_C = k_D$ and $b_C = b_D$, then $s_C = s_D$.

In our context, sibling ids are used to determine siblings in M, and variable bindings are used to determine which elements caused certain sequent firings, while sequent numbers may not be used explicitly. Now, with provenance information above, we will redefine siblings, origin facts, and parents from Definition 2.12 (page 16) and fact ids from Definition 2.2 (page 7).

Definition 2.15. Let M be a model for a geometric theory Σ.

1. *Siblings information* is a function $siblings : \{sids\} \to \mathcal{P}(\{fids\})$, where $siblings(sid)$ consists of each fact $A \in M$ (denoted by sid s_A rather than A) such that $s_A = sid$.

2. *Element origins* are defined as a function $origin : dom(M) \to \{fids\}$ such that for each $e \in dom(M)$, $origin(e) \in U$ is the least fid of any fact in which e appears.

3. *Parents information* is the same as in Definition 2.12 on page 16 and depends on *siblings* and *origin*.

4. *Fact ids* (*fid*s) are defined such that, in addition to Definition 2.2, for any two facts $A, B \in M$, $fid_A < fid_B$ iff $s_A < s_B$, that is, fact ids are grouped by sibling ids. Moreover, fids are assigned consecutively, breaking ties arbitrarily for sibling facts.

With the several definitions above, we are ready to describe the previous core-finding algorithm $\textsc{FindCore}^E$ [11].

Chapter 3

Previous Algorithms

3.1 Outline

As described in the introduction, there are core-finding algorithms by Fagin et al. [5], Gottlob [6], Gottlob and Nash [7, 8], and Pichler and Savenkov [11]. The core-finding algorithm FindCore^E by Pichler and Savenkov [11] is given on the following page. Note that the algorithm here is modified from the original data exchange context so that we already have an initial model (the universal solution) M from the start and that we assume solutions to these theories always exist. Variable names are also changed to better fit the context of this project.[1]

The idea of the algorithm is to iteratively compute a succession of nested proper retracts. The procedure starts with an initial model $U := M$. Then it gradually and repeatedly shrinks the model U until it cannot be any smaller, at which point it has reached the core. In each step of attempting to shrink the current model U, it selects each pair of distinct elements (x, y). Then it computes a sub-model M_{xy} of M, which contains origin facts $origin(x)$ and $origin(y)$ and is closed under parents and siblings. More specifically, $M_{xy} := F_x \cup F_y$ where F_z for $z = x$ and $z = y$ can be computed iteratively using the

[1]For instance, M, M_{xy}, elm are used in place of T^{Σ}, T_{xy}, var.

Procedure FindCore^E ([11]):

Input: Theory Σ
Input: Initial model $M \models \Sigma$
Output: Core of M

```
(1) Set $U := M$
(2) **for each** $x, y \in dom(U), x \neq y$ **do**:
(3)     $M_{xy} := F_x \cup F_y$ by calling ComputeFz with each of $x$ and $y$
(4)     Find $h : M_{xy} \to U$ s.t. $h(x) = h(y)$
(5)     **if** such $h$ exists:
(6)         Extend $h$ to an endomorphism $h'$ on $U$ by calling Extend
(7)         Transform $h'$ into a retraction $r$
(8)         Set $U := r(U)$
(9) **return** $U$
```

Procedure ComputeFz ([11])

Input: Element $z \in dom(T^\Sigma)$
Output: Set F_z

```
(1) Set $F_z := \emptyset$
(2) Set queue $Q := \{origin(z)\}$
(3) **while** $Q$ is not empty **do**:
(4)     Set fact $A := removeFrom(Q)$
(5)     Set $F_z := F_z \cup \{A\}$
(6)     Set $Q := Q \cup (siblings(A) \setminus F_z)$
(7)     Set $Q := Q \cup (parents(A) \setminus F_z)$
(8) **return** $F_z$
```

```
Procedure EXTEND ([11]):

Input: Model $M$
Input: Submodel $W \subseteq M$ closed under parents and siblings
Input: Homomorphism $h : W \to B$ with $B \models \Sigma$
Output: Homomorphism $h' : M \to B$ such that
        $\forall x \in dom(W), h'(x) = h(x)$

(1)  Set $h' := h$
(2)  while exists a fact $A \in M \setminus W$,
         s.t. $parents(A) \neq \emptyset$ and $parents(A) \subseteq W$ do:
(3)      Set $P := parents(A)$
(4)      Set $S := \{A\} \cup siblings(A)$
(5)      Find homomorphism $g : S \cup P \to B$,
             s.t. $\forall x \in dom(g) \cap dom(h') : g(x) = h'(x)$
(6)      Set $h' := h' \cup g$
(7)      Set $W := W \cup S$
(8)  return $h'$
```

algorithm ComputeFz (which is essentially the same as the procedure by Pichler and Savenkov [11] though more concrete). By adding into F_z siblings and parents of any fact that is to be included in F_z, it is obvious that F_z is closed under parents and siblings. Next it tries to find a homomorphism $h : M_{xy} \to U$ which maps x and y to the same element. A naïve approach is used to search for such a homomorphism. If h does not exist, then x and y are not mapped to the same element in the core. Otherwise, h can always be extended to an endomorphism $h' : U \to U$, which is then transformed into a proper retraction r on U. The algorithm finishes the step by applying r on U, resulting in a smaller model $r(U)$ where x and y have been mapped to the same element. The step is executed for each remaining (x, y) until none is left, meaning the core is found and no progress can be made further.

The subroutine Extend extends the initial homomorphism h by iteratively adding facts from M into W while mapping new elements for g such that the new mapping $h' \cup g$ is consistent with the previous mapping h'.

The subroutine relies on the fact that W is closed under parents and siblings because once elements in parents of the sibling set have been mapped, that is, h' is defined on those elements, g can then be defined in a straightforward way for new elements in the sibling set for which h' is undefined. More specifically, chasing Σ to obtain M gives us information about which facts and elements directly caused the creation of certain facts and elements.[2] In other words, extending h' requires no backtracking because of the deterministic nature of Σ (no disjunction). However, we will explore in the Section 3.3 why disjunctions may cause this procedure to fail.

The next part will cover relevant theorems for the correctness and running time of FindCore^E The theorems have been put into the new context as the algorithm early in this section.

3.2 The Correctness of the Algorithm

Lemma 3.1. ([11]) *Let Σ be a weakly acyclic geometric theory and let a model $M \models \Sigma$. For each $x, y \in dom(M)$, the submodel M_{xy} (Definition 2.13) satisfies:*

1. *$origin(x), origin(y) \in M_{xy}$,*
2. *every fact in W was created by firing a non-full sequent,*
3. *$|dom(M_{xy})| \leq b$.*
4. *M_{xy} can be computed in time $O(b^c)$ for some constants b and c which depend only on Σ.*

Theorem 3.2. ([11]) *Let Σ be a weakly acyclic geometric theory, let U be a retract of M, let $x, y \in dom(M)$, and let $M_{xy} \subseteq M$ be computed in Definition 2.13 above. Then there is a constant c, depending only on Σ, such that we can check in time $O(|dom(M)|^c)$ whether there exists a homomorphism*

[2]That information is called *source positions* [11], but it is not used in this project because it is unavailable to us and is replaced by *provenance* information, defined in Subsection 2.2.3.

$h : M_{xy} \to U$, $h(x) = h(y)$.

Theorem 3.3. ([11]) *Let M be a model satisfying a weakly acyclic geometric theory Σ* without disjunctions*. Let model $B \models \Sigma$ and let $W \subseteq M$ be a submodel such that W is closed under parents and siblings over facts and that each fact in W was created by firing a non-full sequent. Then any homomorphism $h : W \to B$ can be extended in time $O(|dom(M)|^b)$ into a homomorphism $h' : M \to B$ such that $\forall x \in dom(h) : h(x) = h'(x)$, where b depends only on Σ.*

Theorem 3.4. ([8]) *Let M be a model and let h be an endomorphism on M such that $h(x) = h(y)$ for some $x, y \in dom(M)$. Then there exists a retraction r on M such that $r(x) = r(y)$, and r can be computed in time $O(|dom(M)|^2)$.*

Theorem 3.5. ([11]) *Let Σ be a weakly acyclic geometric theory without disjunctions and let $M \models \Sigma$. Then* FindCore^E *correctly computes the core of M in time $O(|dom(M)|^b)$ for some b that depends only on Σ.*

3.3 Problem with Disjunctions

Disjunctions in geometric theories complicate the problem of core finding: The algorithm FindCore^E [11], especially the lifting function Extend on page 21, does not always work with such theories due to non-determinism from disjunctions. More specifically, the following two examples demonstrate that Theorem 3.3 does not hold when Σ contains disjunctions, meaning that Theorem 3.5 also no longer holds and that the algorithm FindCore^E requires a fix in order to correctly computes cores when disjunctions are present in theories. The first example is the case where FindCore^E fails to extend an initial homomorphism h, given that h exists for some elements x, y, because there is no extension to an endomorphism h' in the first place. On the other hand, the second example will show that although such extension to the endomorphism h' exists, FINDCORE^E can fail due to making unlucky choices during the incremental extension. Now, consider the following example.

Example 3.6. Let a theory Σ be as follows:

$\sigma_1 : \top \to \exists x.P(x)$
$\sigma_2 : \top \to \exists y.Q(y)$
$\sigma_3 : P(x) \to \exists y.R(x, y)$
$\sigma_4 : Q(x) \to \exists y.R(x, y)$
$\sigma_5 : R(x_1, y) \wedge R(x_2, z) \to x_1 = x_2$
$\sigma_6 : R(x, y) \to \exists z.S(y, z)$
$\sigma_7 : S(y, z) \to A(z) \vee B(z).$

Then one possible initial model M satisfying Σ is

$$\{P(a), Q(a), R(a, b), R(a, c), S(b, d), S(c, e), A(d), B(e)\},$$

where a, b, c, d, and e are the elements in the model M. Note that Σ is weakly acyclic.

Specific details about the theory are as the following: The first four sequents force the existence of two facts of the relation R. Then sequent σ_5 equates the first arguments of these facts, essentially making facts of relations P and Q unimportant since then. The purpose of P and Q is to allow multiple similar facts (redundancy). σ_6 then creates two more facts, one for each fact in R, and finally, σ_7 gives choices for the Chase to make. Let us assume M was created from choosing different branches of σ_7.

To show how this can be problematic to Theorem 3.3, all four conditions of the theorem (except for the disjunctions) must be satisfied while the conclusion must not. Let $(x, y) = (b, c)$, $B = M \models \Sigma$, $W = M_{xy} = M_{bc} = \{P(a), Q(a), R(a, b), R(a, c)\} \subseteq M$, and $h : W \to B$ where $h(a) = a$ and $h(b) = h(c) = c$. Also, $P(a)$ originated from σ_1 and has no sibling or parent. $Q(a)$ is similar except that it came from σ_2. $R(a, b)$ came from σ_3 and has a parent $P(a)$, and $R(a, c)$ came from σ_4 and has a parent $Q(a)$. Thus W

is closed under parents and siblings over facts. All σ_1, σ_2, σ_3, and σ_4 are non-full sequents. Therefore, all the conditions in Theorem 3.3 are satisfied. However, no extension of h to $h' : M \to B$ exists because $h'(d)$ can never be equal to $h'(e)$, but the fact that $h(b) = h'(b) = c$ forces $h'(d) = h'(e) = e$, leading to a contradiction.

In another case, we may find that a particular endomorphism $h : W \to B$ can be extended to $h' : M \to B$, but that during the extension, the algorithm happens to pick an unlucky choice that leads to a dead end in a later step of lifting. Consider the following example.

Example 3.7. Let theory Σ be as follows:

$\sigma_1 : \top \Rightarrow \exists x.P(x)$
$\sigma_2 : \top \Rightarrow \exists y.Q(y)$
$\sigma_3 : \top \Rightarrow \exists z.R(z)$
$\sigma_4 : P(x) \Rightarrow \exists y.S(x, y)$
$\sigma_5 : Q(x) \Rightarrow \exists y.S(x, y)$
$\sigma_6 : R(x) \Rightarrow \exists y.S(x, y)$
$\sigma_7 : Q(x_1) \wedge S(x_1, y) \wedge R(x_2) \wedge S(x_2, z) \Rightarrow x_1 = x_2$
$\sigma_8 : P(x) \wedge Q(y) \Rightarrow P(y)$
$\sigma_9 : S(x, y) \Rightarrow \exists z.T(y, z)$
$\sigma_{10} : T(y, z) \Rightarrow A(z) \vee B(z)$.

And let the initial model M be

$$\{P(a_1), P(b_1), Q(b_1), R(b_1), S(a_1, a_2), S(b_1, b_2), S(b_1, c_2),$$

$$T(a_2, a_3), T(b_2, b_3), T(c_2, c_3), A(a_3), B(b_3), A(c_3)\}.$$

Notice that M consists of three "trails" of facts, with the a-trail separated from the rest. The heads of b- and c-trails are joined by the equality in σ_7 and consist of $P(b_1)$ and $Q(b_1)$ not in the a-trail; and the tail of the b-trail, ending with the fact $B(b_3)$ of relation B, is distinct from other tails, ending with facts of relation A. Thus, the only possible collapse of trails within M is from the a-trail to the c-trail, i.e., a retraction r: $r(a_1) = b_1$, $r(a_2) = c_2$, $r(a_3) = c_3$, and for each $x \in dom(M) \setminus \{a_1, a_2, a_3\}$, $r(x) = x$.

Again, first let us satisfy the premise of Theorem 3.3. Let $(x, y) = (a_1, b_1)$, $B = M \models \Sigma$, $W = M_{xy} = M_{ab} = \{P(a_1), Q(b_1)\} \subseteq M$, and $h : W \to B$ where $h(a_1) = h(b_1) = b_1$. Since $P(a_1)$ and $Q(b_1)$ originated from σ_1 and σ_2 (respectively) and have no sibling or parent, W is closed under parents and siblings. All σ_1, σ_2, σ_3, and σ_4 are non-full sequents. Therefore, all the conditions in Theorem 3.3 are satisfied.

However, the original algorithm does not handle this case well. There are two choices for a_2 to map, b_2 and c_2. As note earlier, the second can lead to the retraction r, but without any predictive method, let us assume that the algorithm blindly chooses the first choice in the lifting: $h'(a_2) = b_2$. Then we are forced to map $a_3 \to b_3$, but since $A(b_3) \notin M$, we reach a dead end as in Example 3.6 that is impossible in the data exchange setting.

From Examples 3.6 and 3.7, we state the following proposition.

Proposition 3.8. FINDCORE^E *[11], especially* EXTEND, *may fail when theories contain disjunctions. In other words, Theorems 3.3 and 3.5 no longer hold for those theories.*

Chapter 4

The Revised Algorithm

4.1 Outline for the Revised Algorithm

As we switch contexts from data exchange to the more general setting, some properties of the core-finding problem change while others remain the same. More specifically, the following theorems stated in Section 3.2 still apply here:

- Lemma 3.1 about M_{xy} (M_{xy} will still be used in the algorithm),
- Theorem 3.2 about finding an initial homomorphism $h : M_{xy} \to U$ ($U \subseteq M$), and
- Theorem 3.4 about the existence of proper retractions given proper endomorphisms.

However, as we showed in Section 3.3 above, Theorem 3.3 for lifting generally does not hold for theories with disjunctions, our more general setting. As such, Theorem 3.5 no longer applies, since it depends on that theorem. Moreover, the closure under retraction of geometric theories needs to be extended to theories containing disjunctions (see Theorem 2.5 on page 12).

This chapter is structured in the following way: First we describe element signatures and the signature testing. Next we describe how we revised the previous algorithm [11] and which features are changed or added. Finally, we prove for the correctness of the algorithm.

In addition, we can summarize ideas about the revision of the algorithm as follow:

1. We fix the previous algorithm by switching to a slower method when the original fails to extend in some loop of FindCore^E. This method guarantees correctness but can be exponentially slower than before, since it is much more naïve than the original. Thus, the revised algorithm may not run in polynomial time in the number of elements in the worst case.

2. We use a heuristic called "signature testing" to prune homomorphism searching space. It is both to improve the running time of the algorithm in general and to remedy the exponential time of the fix in Idea #1. To implement this heuristic, each element will have a signature, or a fingerprint, which contains partial important information about that element. That information, which is analogous to but more meaningful than hash values, will allow us to quickly determine whether two elements can be mapped to one another in any homomorphism, though possibly with false positives. The element signature will be defined in the next section.

3. One significant difference of between FindCore^E and our algorithm is the input information available to the algorithms. FindCore^E can use the chase step sequence [11], which contains complete event information of each sequent that was fired, by which facts and which elements, from start to end. On the other hand, we are not given that information directly.[1] Instead, we have "provenance information," (already defined in Subsection 2.2.3 on page 17) which is roughly a subset of the sequence. More precisely, it is missing complete theory information

[1]from Hominy, a model finder from which we obtain models

and original elements before firing equalities. Therefore, there will be differences in using information and in the implementation, although the concept remains intact.

4.2 Element Signature

Definition 4.1. Fix a language $\mathcal{L}$. Let M be a model. For each element $e \in dom(M)$, the *signature* $Sig(e)$ consists of, for each relation symbol $P \in \mathcal{L}$, a set $Sig_P(e)$ of sets of indices where each set $s = \{i_1, i_2, \ldots, i_k\} \in Sig_P(e)$ if there exists a fact $P(a_1, a_2, \ldots, a_n), n \geq k$, such that for each $i \in \{1, 2, \ldots, n\}$, $a_i = e \iff i \in s$. Note that an indices set s may come from more than one fact, but which fact specifically is irrelevant.

With signatures defined, we can state the following definitions, which rely on signatures.

Definition 4.2. Fix a language $\mathcal{L}$. Let M be a model. The (initial) *signature-compatible pairs set* (*scp*) scp_1 is a set consisting of all element pairs (x, y) where $x, y \in dom(M)$ and, for each relation symbol $P \in \mathcal{L}$ and each indices set $s_x \in Sig_P(x)$, there exists $s_y \in Sig_P(y)$ such that $s_x \subseteq s_y$. In this case, we call (x, y) *a compatible pair.*

In fact, the scp is the only thing we use in signature testing, since the set apparently tells us which elements to which certain elements can map. However, note that this set is not fixed throughout the algorithm. Some pairs (a, b) may be removed as we discover later that they are false positives, i.e., no valid endomorphism h on M with $h(a) = h(b) = b$ (more about this later in Section 4.3). In any case, we can use the scp to determine which elements are rigid in the current model.

Definition 4.3. Given *scp* for a model M, each element $e \in dom(M)$ is *rigid* w.r.t. *scp* if $(e, e) \in scp$ and $(e, x) \in scp$ implies $x = e$. In other words, (e, e) is the only pair in *scp* with the left element being e.

```
Procedure FindCore$^D$:

Input: Theory $\Sigma$
Input: Initial model $M \models \Sigma$
Output: Core of $M$

(1)  Set $U := M$
(2)  Set $scp := \{(x, y) \mid x, y \in dom(M) \wedge Sig(x) \subseteq^\star Sig(y)\}$
(3)  for each $(x, y) \in scp, x \neq y$ do:
(4)      $M_{xy} := F_x \cup F_y$ by calling ComputeFz with each of $x$ and $y$
(5)      Find $h : M_{xy} \to U$ s.t. $h(x) = h(y) = y$
(6)      if such $h$ exists:
(7)          Extend $h$ to an endomorphism $h'$ on $U$ by calling Extend$^D$
(8)          if Extend$^D$ returned FAILURE:
(9)              Find $h' : M \to U$ s.t. $h'(x) = h'(y) = y$
(10)         if $h'$ is well defined for $(x, y)$:
(11)             Transform $h'$ into a retraction $r$
(12)             Set $U := r(U)$
(13)             Set $scp := \{(x, y) \mid (x, y) \in scp \wedge x, y \in dom(U)\}$
(14) return $U$
```

One advantage of a shrinking scp is that we have smaller search space. A notion of rigidity with scps will also help improve the running time of the algorithm.

4.3 Revised Algorithm (FindCore^D)

This section will describe ideas behind the algorithm (FindCore^D) in details, including preprocessing.

```
Procedure EXTEND$^D$:

Input: Model $M$
Input: Submodel $W \subseteq M$ closed under parents and siblings
Input: Homomorphism $h : W \to B$ with $B \models \Sigma$
Output: Homomorphism $h' : M \to B$ such that
        $\forall x \in dom(W), h'(x) = h(x)$,
        or returns FAILURE

(1)  Set $h' := h$
(2)  **while** exists a fact $A \in M \setminus W$,
         s.t. $parents(A) \neq \emptyset$ and $parents(A) \subseteq W$ **do**:
(3)      Set $P := parents(A)$
(4)      Set $S := \{A\} \cup siblings(A)$
(5)      Find homomorphism $g : S \cup P \to B$,
             s.t. $\forall x \in dom(g) \cap dom(h') : g(x) = h'(x)$
(6)      **if** such $g$ does not exist:
(7)          **return** FAILURE
(8)      Set $h' := h' \cup g$
(9)      Set $W := W \cup S$
(10) **return** $h'$
```

4.3.1 Preprocessing

Provenance to model

Each fact described in provenance is included in $U_1 = M$ (the initial retract) along with fact ids (*fids*) and sibling ids (*sids*). Facts are assigned fids in increasing order of sids, breaking ties arbitrarily for siblings with the same sid. Note that many facts may share the same fact values, but they are distinguished by fids.

Model information

Crucial information about models used in core computation includes *siblings*, *element origins*, and *signature-compatible pairs* (*scp*s). Fix a language $\mathcal{L}$. Siblings information is a function $siblings_1 : \{sids\} \to \mathcal{P}(\{fids\})$, where $siblings_1(sid)$ consists of all facts (denoted by fids) in U_1 whose sids equal sid. Element origins are defined as a function $origin_1 : dom(U_1) \to \{fids\}$ such that for each $e \in dom(U_1)$, $origin_1(e) \in U_1$ is the least fid of any fact in which e appears. Finally, the scps scp_1 is a set consisting of all element pairs (x, y) where $x, y \in dom(U_1)$ and, for each relation symbol $P \in \mathcal{L}$ and $s_x \in Sig_P(x)$, there exists $s_y \in Sig_P(y)$ such that $s_x \subseteq s_y$ (the meaning of $\subseteq^\star$ in line (2) of FindCore^D). That is, (x, y) is a compatible pair.

4.3.2 Main Core-Finding Function

Given an initial model U_0, along with its information about siblings, origins, and scps, we compute its core by repeatedly executing a loop to gradually shrink the model. In the ith loop ($i \geq 1$), we pick some $(x, y) \in scp_i, x \neq y$; if there is no such (x, y), we stop and have found the core $M_c = U_i$. Each loop consists of a sequence of the following steps:

Computing M_{xy}

For each element $z \in \mathit{dom}(U_i)$, let F_z denote the smallest set that contains $\mathit{origin}_i(z)$ and is closed under parents and siblings. We construct F_z incrementally using the procedure ComputeFz on page 20 as follows: first let $F_z := \{\mathit{origin}_i(z)\}$, then for any fact in F_z, add to the set its siblings and parents. Repeat the process until the set reaches a fixed point. Finally, let $M_{xy} := F_x \cup F_y$.

Initial homomorphism

The idea is to find a non-injective homomorphism $h : M_{xy} \to U_i$ where $h(x) = h(y) = y$. We consider each non-rigid block B in M_{xy} and define h independently. If $x \in \mathit{dom}(B)$ or $y \in \mathit{dom}(B)$, then we can use exhaustive search with signature testing and partial-answer checking (checking if the partial h is a homomorphism from some subset of M_{xy} to U_i as we build h) to compute h restricted to B in $O(|\mathit{dom}(B)|^c)$ for some c that depends only on Σ (by Theorem 3.2). Otherwise, if $x, y \notin \mathit{dom}(B)$, we can use the previous retract $r_{i-1} : M \to U_i$ (r_0 is the identity) to define h on block B, that is, we set $h(z) = r_{i-1}(z)$ for each $z \in \mathit{dom}(B)$, since h is not required to be non-injective except for $h(x) = h(y) = y$. Additionally, we define $h(z) = z$ for each rigid element (as if constant) $z \in \mathit{dom}(M_{xy})$. If such $h : M_{xy} \to U_i$ does not exist, then no $h' : M \to U_i$ where $h'(x) = h'(y)$ exists either, and therefore (x, y) can be removed from scp_i (i.e., we set $\mathit{scp}_{i+1} := \mathit{scp}_i \setminus \{(x, y)\}$) because no mapping exists such that $h'(x) = y$. But if such h exists, then there *may* be an extension $h' : M \to U_i$ of h where $h'(x) = h'(y)$ and for each $z \in \mathit{dom}(T_{xy})$, $h(z) = h'(z)$. Theorem 3.3 guarantees the existence of such h' for weakly acylic geometric theories without disjunctions, while Proposition 3.8 shows that it is always not the case for theories with disjunctions.

Note that in FindCore^E [11], $h(x) = h(y)$ is not forced to be equal to y, but it is the case here because signatures are used to eliminate mapping possibilities and Lemma 4.4 will prove the validity of the method.

Extend

This is the procedure EXTEND^D on page 31. Let W be the current submodel, where $M_{xy} \subseteq W$, there exists a homomorphism $h : W \to U_i$, and W is closed under parents and siblings. Initially let $W := M_{xy}$ and h denote the initial homomorphism found in the previous step. Unlike EXTEND [11], which uses "source positions" to map each new fact outside of W, we select a set of siblings S to be mapped group by group, assuming the parents of facts in S are already in W. In fact, if we select S in increasing order of sids, then that assumption can be made. If S has an sid less than all facts in W, then facts in S do not have a parent and thus can be added to W safely (and update h as appropriate). The procedure to update h to $h : (W \cup S) \to U_i$, considering S to be a single block (even if it consists of multiple non-rigid blocks or it is part of larger blocks), is the same naïve approach used in the previous step. This procedure is repeated until $W = U_i$. Since the number $|dom(S)|$ is bounded by some constant c, each small extension step runs in polynomial time, $O(|dom(U_i)|^c)$.

Fix for Extend

However, just as EXTEND, EXTEND^D does not well handle models for theories with disjunctions (Proposition 3.8), but now it reports failure to the caller FINDCORE^D when it cannot find such an extension to an endomorphism h'. That does not mean that such h' does not exist at all, but rather it is "too hard" to compute. If failure is reported, then FINDCORE^D will attempt to construct $h' : M \to U_i$ from scratch, using signature testing to prune search. The algorithm can proceed if it successfully find h'; otherwise, as in the part about initial homomorphisms, there is no such h' in the first place. Clearly, this method can avoid failure unlike the original FINDCORE, but the drawback is that the algorithm no longer runs in polynomial time in terms of $|dom(M)|$.

Transformation to retraction

In this step, we transform the non-injective endomorphism $h' : M \to U_i$, where $h'(x) = h'(y) = y$, into a non-injective retraction $r : M \to U_i$, such that $r(x) = r(y) = y$. The procedure can be summarized as follows: find cycles and paths, compute their lengths, and use exponentiation by squaring to repeatedly compose h' into r. By Theorem 3.4, such r always exists given the h'.

Applying retraction to model

If there exists a non-injective retraction $r_i : M \to U_i$, then we construct the new model U_{i+1} and its information as follows:

- **Model**: $U_{i+1} := \{\text{fact } A \in U_i \mid r_i(A) = A\}$ (facts in the image of the retraction r_i; therefore, $r_i : M \to U_{i+1}$).
- **Siblings information**: Siblings remain the same since we always try to find a retraction r from the initial model M.
- **Element origins**: Similarly to siblings information, element origins remain the same.
- **Signature-compatible pairs**: scp_{i+1} contains all pairs $(x, y) \in scp_{i+1}$ such that $x, y \in dom(U_{i+1})$.

Otherwise, let $scp_{i+1} := scp_i \setminus \{(x, y)\}$ (since we now know that there does not exists a mapping that maps x to y in M) and leave everything else the same, that is, $U_{i+1} := U_i$ for the new model and $r_i := r_{i-1}$.

4.4 Correctness of the Revised Algorithm

The following lemma shows that signature testing does not exclude potential possibilities of mappings that are required to reach the core.

Lemma 4.4. *(Signature-testing correctness) Let Σ be a geometric theory and let $M \models \Sigma$. For each (x, y) where $x, y \in dom(M)$ and $x \neq y$, if there exists an endomorphism h on M such that $h(x) = h(y) = y$, then $(x, y) \in Sig_M$.*

Proof. Let us assume $(x, y) \notin Sig_M$ for chosen elements $x, y \in dom(M)$. This means that, according to the signature, there is some fact $P(\vec{a}) \in M$ for some relation symbol P and tuple $\vec{a} \in dom(M)$, but there does not exist a fact $P\left(\vec{b}\right) \in M$ for any tuple $\vec{b}$, such that for each i, $(a_i = x)$ implies $(b_i = y)$. Suppose that there exists an endomorphism $h : M \to M$ such that $h(x) = h(y) = y$. Then $P(h(\vec{a})) \in M$, but since there is no such $\vec{b}$ as described above, $P(h(\vec{a})) \notin M$, a contradiction. Therefore, there does not exist any endomorphism h on M such that $h(x) = h(y) = y$. □

Lemma 4.5. *Let M be a model for a geometric theory and let U be a retract of M. For any $x, y \in dom(M)$, if $h' : M \to U$ exists such that $h(x) = h(y) = y$, then executions during lines (5)-(10) of* FindCore^D *will correctly compute h', and if h' does not exist, the algorithm will report that correctly.*

Proof. This proof relies partially on Theorem 3.3 at least for the non-disjunction case. Although Proposition 3.8 states that the theorem does not hold in general, if Extend can extend $h : M_{xy} \to U$ to h', then h' is computed as required. Also, since line (9) checks for every possible endomorphism (regardless of whether we use signature testing, because it only eliminate non-homomorphic mappings), the algorithm will always find one such h' if it exists. In general, there are three cases for h and h':

1. h does not exist: This means that h' cannot exist and that FindCore^D does not continue to line (7). Therefore, it correctly reports that h' does not exist.

2. h exists, but h' does not: This guarantees that line (7) will return with an error, but line (9) cannot find h' either, because it does not exists. Thus, the algorithm still correctly reports that h' does not exist.

3. Both h and h' exist: If line (7) successfully finds one such h' from lifting h, then we have found one. Otherwise, if Extend^D returns failure,

then line (9) will search exhaustively for one. As stated above, the algorithm will always find one such h' if it exists. Therefore, the algorithm correctly computes h', which exists.

Overall, lines (5)-(10) of FindCore^D correctly compute an endomorphism h' if h' exists, and report correctly that h' does not exist, otherwise. $\square$

Theorem 4.6. *Let Σ be a weakly acyclic geometric theory and let $M \models \Sigma$. Then* FindCore^D *correctly computes the core of M.*

Proof. The proof of this theorem is based on Theorem 3.5 ([11]) but with several modifications. The constraint $(x, y) \in scp$ in line (3) is sufficient by Lemma 4.4. The restriction on h in line (5) is justified by Lemma 2.6. The correctness of execution during lines (5)-(10) is justified by Lemma 4.5, and for the correctness of execution during lines (12)-(13), Theorem 2.5. As stated about provenance information in Section 4.1, original elements in a model before they were equated are lost. Thus the worst thing that can happen because of the lost information is that blocks are merged and result in unbounded block size, but that should not affect the correctness of the algorithm. $\square$

Note that this new algorithm does not run in polynomial time in terms of $|dom(M)|$ in the worst case mostly because of the execution of line (9). Also, the unbounded block size mentioned in the proof may cause the exponential homomorphism-searching time, although does not matter since line (9) runs in exponential time in the worst case anyway.

Chapter 5

Evaluation

5.1 Implementation

As mentioned in the introduction, the FindCore^D algorithm was implemented in Haskell. The program requires an input from Hominy, a geometric model-finding tool. Thus we linked our algorithm with Hominy for convenience, for input models can be passed directly and easily from Hominy to the core-finding program.

5.2 Experiments

We ran two experiments on randomly generated geometric theories:

1. executing the revised algorithm with signature testing (Algorithm A), which is the default option, and
2. comparing the algorithm that uses signature testing (Algorithm A) and the one without the heuristic (Algorithm B).

In both experiments, we used a testing tool in Haskell called QuickCheck to randomly generate theories based on provided random parameters, such as the number of sequents in a theory, and, for each sequent, how likely certain term types will appear when growing formulae top-down. For simplicity, the language $\mathcal{L}$ for theories is fixed, having no more than eight relation symbols, each with arity at most three. Note that these theories excluded equalities.

Moreover, to allow for models with a large number of elements and facts, we randomly created several facts and elements in a model (as if there were a special sequent to create these facts when fired) before letting Hominy build models from it to satisfy the random theory. Otherwise, there would not be enough facts or elements in the model (an input to our algorithm) that satisfies the theory, or, in another case, if we tried to create a theory that is too complicated, then Hominy might take more than several minutes, or forever, to produce even a single model, perhaps because the random theories were not weakly acyclic.

One remark about having Hominy start chasing from a random, non-empty model is that those initial facts and elements do not naturally come from multiple sequent firings but just one, an imaginary one, and that they are "flat," i.e., as if created all at the same time by a non-chase-based model finder). As a result, such a model will be harder to compute for its core, but fortunately, that means we can stress-test our implementation.

Each experiment consists of several test runs. In each run, we obtained only first five (or fewer) models from Hominy and then ran $\textsc{FindCore}^D$ on each model, with the time limit of ten minutes per run, not per model. For Experiment 1, As soon as Algorithm A finished computing the core of a model in any run, the model was included in the results. For the comparison Experiment 2, both algorithms had to finish within the time limit in total for a model to be used. Additionally, since Algorithm B was expected to be significantly slower than Algorithm A, test models in Experiment 2 were easier and faster to than run than those in Experiment 1. Figures below will clearly reflect this difference. Note that data for Algorithm A in Experiment 2 are also included in Experiment 1.

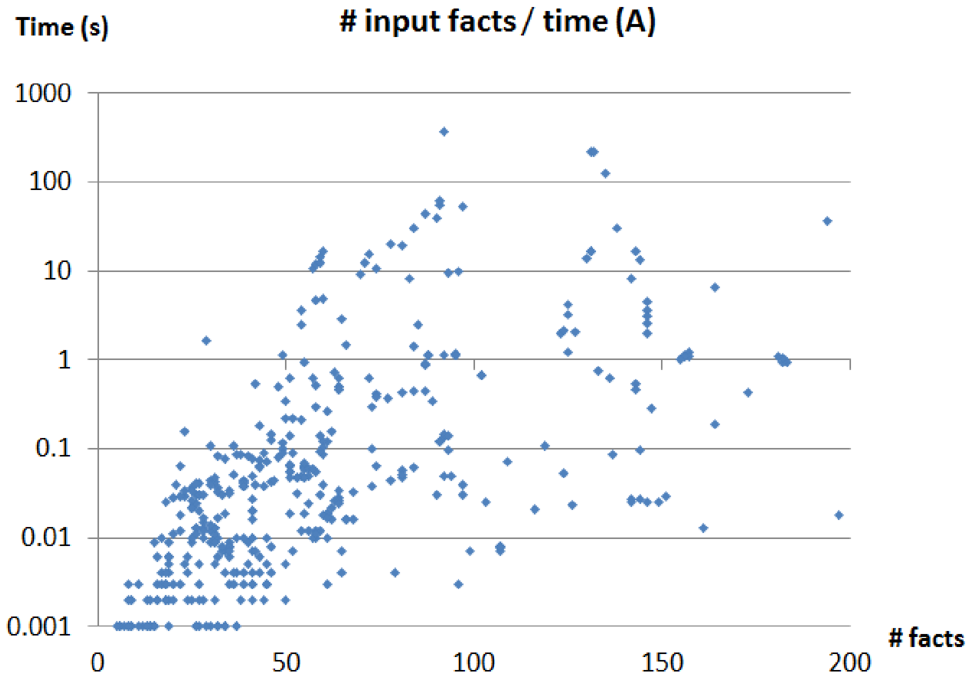

Figure 5.1: Input model size (#facts) vs. running time of Algorithm A

As for other environments and factors, these experiments were run on a 64-bit Windows machine with a quad-core CPU (2.50Ghz) and 8GB RAM. During the runs, the computer was left unused just to run the experiments.

5.3 Results

Experiment 1

Figure 5.1 shows the running time of Algorithm A (with signature testing) in terms of the number of facts in a model. Each dot represents one test model, and there are 476 models in total. The y-axis is in log scale because of high variation of values. Note that some models are excluded because the algorithm did not finish in time, and a few models have more than 200 facts and

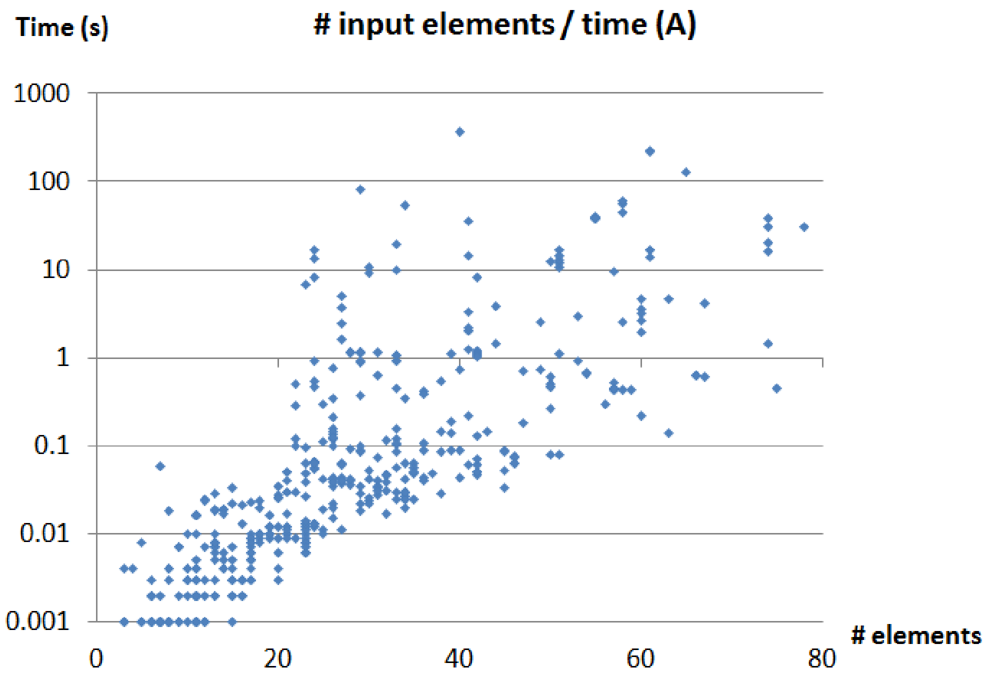

Figure 5.2: # Input elements vs. running time of Algorithm A

thus are not shown. In the figure, notice that for most models, the algorithm took less than one second, possibly because of the data from Experiment 2. Notice also that the running time generally increases (exponentially) as the model size increases (linearly).

Figure 5.2 also shows the running time of Algorithm A but in terms of the number of elements in a model. Only a few models are excluded from the plot because they have more than 80 elements. Notice that there is a clearer trend line than in Figure 5.1, so the running time is approximately exponential in the number of elements. One reason is that the input models for Hominy were randomly generated and not initially empty. Notice also that there are two separate groups in this figure, a dense group below the 0.1 second and a sparse group covering the rest of the plot. It turns out that the first group are models from Experiment 2 while the second group is from

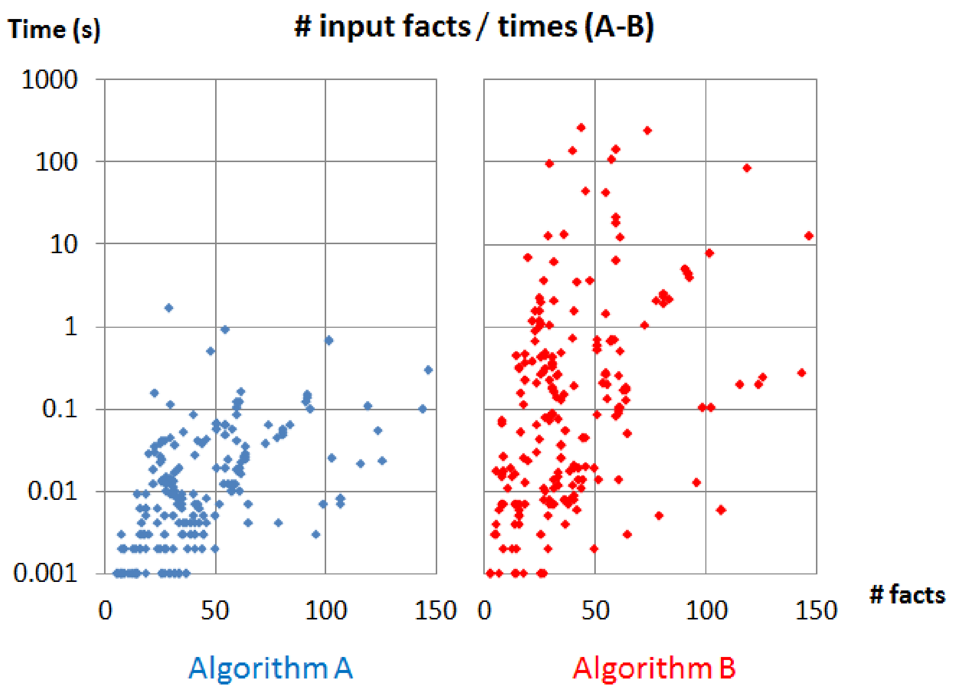

Figure 5.3: Comparison of running time of Algorithms A and B (model size)

harder models only used for this experiment.

Experiment 2

Figure 5.3 compares the running time of Algorithm A with that of Algorithm B. The x-axis is the number of facts in a model, and the y-axis is time in seconds, still in log scale. There are 239 models in this experiment, and similarly to Experiment 1, a few outliers are excluded to show the current plots. What is clear from this comparison is that signature testing used in Algorithm A greatly reduces running time, with a factor of number as high as 100. (Moving up one horizontal line means ten times slower.) In fact, for many models, Algorithm A did not need to do anything because element signatures reported that no two elements were compatible, and thus those models were already cores.

Chapter 6

Future Work

There are several questions and suggestions for improvement left to be done: about algorithms, heuristics, and optimizations.

We have two questions specifically about the core-finding algorithm. The first one involves M_{xy} (i.e., T_{xy} [8, 11]) and finding an initial homomorphism h which guarantees the existence of an extension from it. What we have now is to replace M_{xy} with the entire model M, and thus there is no need to extend h because h is already an endomorphism, but the drawback is an exponential search time. We wonder if there is some slightly larger submodel M'_{xy}, for any $x, y \in dom(M)$, such that $M_{xy} \subseteq M'_{xy}$ and every homomorphism $h : M'_{xy} \to M$ can be extended to $h' : M \to M$. If M'_{xy} is not significantly larger than M_{xy} (same for their block size), then we will have a polynomial-time algorithm for finding cores for geometric theories with disjunctions.

The second question is whether it is possible to avoid extending homomorphisms to the initial model M in every loop. More precisely, can we only extend $h : W \to U$, such that $W \subseteq U$, to $h' : U \to U$ where $U \subset M$ and then consider U to be the new initial model of the next loop? Intuitively, M contains all information necessary to construct M_{xy}, for elements $x, y \in dom(M)$, but in general $M_{xy} \nsubseteq U$. Also this approach, if it can be used, may significantly improve the running time.

Since this project used only one heuristic, we wonder what other heuristics can be useful. Perhaps, comparisons and combinations among different heuristics can be a focus of study. We have an idea about a heuristic to try, which involves looking up facts in a model. When checking whether a mapping between models is a homomorphism, we have to determine whether each fact whose elements are mapped is in the destination model. Especially, when mappings are partial, looking up a partially mapped fact is like a database query. Therefore, we wonder if using indexing techniques from database, as a heuristic, will reduce time for looking up facts and finding homomorphisms.

Bibliography

[1] Aho, A. V., Beeri, C., and Ullman, J. D. The theory of joins in relational databases. *ACM Trans. Database Syst. 4*, 3 (Sept. 1979), 297–314.

[2] Beeri, C., and Vardi, M. Y. A proof procedure for data dependencies. *J. ACM 31*, 4 (Sept. 1984), 718–741.

[3] Deutsch, A., and Tannen, V. Reformulation of xml queries and constraints. In *Proceedings of the 9th International Conference on Database Theory* (London, UK, UK, 2002), ICDT '03, Springer-Verlag, pp. 225–241.

[4] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. Data exchange: Semantics and query answering. In *Proceedings of the 9th International Conference on Database Theory* (London, UK, UK, 2002), ICDT '03, Springer-Verlag, pp. 207–224.

[5] Fagin, R., Kolaitis, P. G., and Popa, L. Data exchange: Getting to the core. *ACM Trans. Database Syst. 30*, 1 (Mar. 2005), 174–210.

[6] Gottlob, G. Computing cores for data exchange: New algorithms and practical solutions. In *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (New York, NY, USA, 2005), PODS '05, ACM, pp. 148–159.

[7] Gottlob, G., and Nash, A. Data exchange: Computing cores in polynomial time. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (New York, NY, USA, 2006), PODS '06, ACM, pp. 40–49.

[8] Gottlob, G., and Nash, A. Efficient core computation in data exchange. *J. ACM 55*, 2 (May 2008), 9:1–9:49.

[9] Hell, P., and Nesetril, J. The core of a graph. *Discrete Mathematics 109*, 1-3 (1992), 117 – 126.

[10] Maier, D., Mendelzon, A. O., and Sagiv, Y. Testing implications of data dependencies. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 455–469.

[11] Pichler, R., and Savenkov, V. Towards practical feasibility of core computation in data exchange. *Theoretical Computer Science 411*, 7-9 (2010), 935 – 957.