Generating Minimal Models for Geometric Theories

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

by

....

Michael Ficarra

on

30th September, 2010

....

Daniel Dougherty
professor, project advisor

Abstract

This paper describes a method, referred to as the chase, for generating jointly minimal models for a geometric theory. A minimal model for a theory is a model for which there exists a homomorphism to any other model that can satisfy the theory. These models are useful in solutions to problems in many practical applications, including but not limited to firewall configuration examination, protocol analysis, and access control evaluation. Also described is a Haskell implementation of the chase and its development process and design decisions.

Table of Contents

1 Introduction

Introductory text...

1.1 Goals

1.2 The Chase

2 Technical Background

In this paper, possibly ambiguous or uncommon notation will be used and thusly must be clearly defined. Also, topics that are necessary prerequisites will be summarized.

2.1 Models

A *model* $\mathbb{M}$ is a construct that consists of:

- a set, referenced as $|\mathbb{M}|$, called the *universe* or *domain* of $\mathbb{M}$
- a set of pairings of a *predicate* and a non-negative integral arity
- for each predicate R with arity k, a relation $R_k^{\mathbb{M}} \subseteq |\mathbb{M}|$

It is important to distinguish the predicate, which is just a symbol, from the relation that it refers to when paired with its arity. The relation itself is a set of tuples of elements from the universe.

2.2 First-order Logic

First-order logic, also called *predicate logic*, is a formal logic system that is an extension of propositional logic. For our purposes, this logic system will not contain any constant symbols or function symbols, which are commonly included in first-order and propositional logic.

A first-order logic formula is defined inductively by

- if R is a relation symbol of arity k and each of $t_0, \ldots, t_{k-1}$ is a variable, then $R[t_0, \ldots, t_{k-1}]$ is a formula, specifically an *atomic formula*
- $\top$ and $\bot$ are formulæ
- if α is a formula then $(\neg\alpha)$ is a formula
- if α and β are formulæ then $(\alpha \wedge \beta)$ is a formula
- if α and β are formulæ then $(\alpha \vee \beta)$ is a formula
- if α and β are formulæ then $(\alpha \rightarrow \beta)$ is a formula
- if α is a formula and x is a variable then $(\forall x : \alpha)$ is a formula
- if α is a formula and $\vec{x}$ is a set of variables of size k then $(\forall \vec{x} : \alpha)$ is $(\forall x_0 \ldots \forall x_{k-1} : \alpha)$

- if α is a formula and x is a variable then $(\exists x : \alpha)$ is a formula
- if α is a formula and $\vec{x}$ is a set of variables of size k then $(\exists \vec{x} : \alpha)$ is $(\exists x_0 \dots \exists x_{k-1} : \alpha)$

A shorthand notation may sometimes be used which omits either the left or right side of an implication and implies a tautology $(\top \rightarrow \sigma)$ and a contradiction $(\sigma \rightarrow \bot)$ respectively.

Examples of first-order logic formulæ

reflexivity	$\rightarrow R[x, x]$
symmetry	$R[x, y] \rightarrow R[y, x]$
transitivity	$R[x, y] \wedge R[y, z] \rightarrow R[x, z]$
	$\forall\ x : R[x] \vee Q[x]$

2.3 Positive Existential Form

Formulas in *positive existential form* are constrained to using only conjunctions ($\wedge$), disjunctions ($\vee$), existential quantifications ($\exists$), tautologies ($\top$), contradictions ($\bot$), and relations to construct logic expressions.

Though formulæ in positive existential form may at first appear to be quite restrictive, there exists some simple logical tricks to allow more expressiveness. Negation of a relation R with arity k can be implemented by assuming another relation R' with arity k, adding two formulæ to the theory of the form $R \wedge R' \rightarrow \bot$ and $\top \rightarrow R \vee R'$, and using R' where $\neg R$ would be used.

2.4 Geometric Logic

Geometric logic formulæ are implicitly universally quantified implications of positive existential formulæ. A set of geometric logic formulæ is called a *geometric theory*.

explain why GL is useful to us... maybe?

2.5 Variable Binding

The set of free variables in a formula is defined inductively as follows

- any variable occurance in an atomic formula is a free variable
- the free variables in $\top$ and $\bot$ are $\emptyset$
- the free variables in $\neg\alpha$ are the free variables in α

- the free variables in $\alpha \wedge \beta$ are the union of the set of free variables in α with the set of free variables in β
- the free variables in $\alpha \vee \beta$ are the union of the set of free variables in α with the set of free variables in β
- the free variables in $\alpha \rightarrow \beta$ are the union of the set of free variables in α with the set of free variables in β
- the free variables in $\forall x : \alpha$ are the free variables in α that are not x
- the free variables in $\exists x : \alpha$ are the free variables in α that are not x

A *sentence* is a formula with an empty set of free variables.

2.6 Environment

An *environment* for a model $\mathbb{M}$ is a function from a variable to an element in $|\mathbb{M}|$. The syntax $l_{[v \mapsto v']}$ defines an environment $l'(x)$ that returns v' when $x = v$ and returns $l(x)$ otherwise.

2.7 Satisfiability

A model $\mathbb{M}$ is said to satisfy a formula σ in an environment l when

- σ is a relation symbol R and $R[l(a_0), \ldots, l(a_n)] \in \mathbb{M}$ where a is a set of variables
- σ is of the form $\neg\alpha$ and $\mathbb{M} \not\models_l \alpha$
- σ is of the form $\alpha \wedge \beta$ and both $\mathbb{M} \models_l \alpha$ and $\mathbb{M} \models_l \beta$
- σ is of the form $\alpha \vee \beta$ and either $\mathbb{M} \models_l \alpha$ or $\mathbb{M} \models_l \beta$
- σ is of the form $\alpha \rightarrow \beta$ and either $\mathbb{M} \not\models_l \alpha$ or $\mathbb{M} \models_l \beta$
- σ is of the form $\forall x : \alpha$ and for every $x' \in |\mathbb{M}|$, $\mathbb{M} \models_{l[x \mapsto x']} \alpha$
- σ is of the form $\exists x : \alpha$ and for at least one $x' \in |\mathbb{M}|$, $\mathbb{M} \models_{l[x \mapsto x']} \alpha$

This is denoted as $\mathbb{M} \models_l \sigma$ and read "σ is true in $\mathbb{M}$". The notation $\mathbb{M} \models \sigma$ (no environment specification) means that either, under any environment l, $\mathbb{M} \models_l \sigma$.

A model $\mathbb{M}$ satisfies a set of formulæ Σ if for every σ such that $\sigma \in \Sigma$, $\mathbb{M} \models \sigma$. This is denoted as $\mathbb{M} \models \Sigma$ and read "$\mathbb{M}$ is a model of Σ".

2.8 Entailment

A set of formulan Σ is said to *entail* a formula σ ($\Sigma \models \sigma$) if the set of all models satisfied by Σ is a subset of the set of all models satisfied by σ.

The notation used for satisfiability and entailment is very similar, in that the operator used ($\models$) is the same, but they can be distinguished by the type of left operand.

2.9 Homomorphisms

A *homomorphism* from $\mathbb{A}$ to $\mathbb{B}$ is a function $h : |\mathbb{A}| \to |\mathbb{B}|$ such that, for each relation symbol R and tuple $\langle a_0, \ldots, a_n \rangle$ where $a \subseteq |\mathbb{A}|$, $\langle a_0, \ldots, a_n \rangle \in R^{\mathbb{A}}$ implies $\langle h(a_0), \ldots, h(a_n) \rangle \in R^{\mathbb{B}}$.

A homomorphism h is also a *strong homomorphism* if, for each relation symbol R and tuple $\langle a_0, \ldots, a_n \rangle$ where $a \subseteq |\mathbb{A}|$, $\langle a_0, \ldots, a_n \rangle \in R^{\mathbb{A}}$ if and only if $\langle h(a_0), \ldots, h(a_n) \rangle \in R^{\mathbb{B}}$.

The notation $\mathbb{M} \preceq \mathbb{N}$ means that there exists a homomorphism $h : \mathbb{M} \to \mathbb{N}$. The identity function is a homomorphism from any model $\mathbb{M}$ to itself. Homomorphisms are transitive, so $\mathbb{A} \preceq \mathbb{B} \wedge \mathbb{B} \preceq \mathbb{C}$ implies $\mathbb{A} \preceq \mathbb{C}$. However, $\mathbb{M} \preceq \mathbb{N} \wedge \mathbb{N} \preceq \mathbb{M}$ does not imply that $\mathbb{M} = \mathbb{N}$.

Given models $\mathbb{M}$ and $\mathbb{N}$ where $\mathbb{M} \preceq \mathbb{N}$ and a formula in positive-existential form [1] σ, if $\mathbb{M} \models \sigma$ then $\mathbb{N} \models \sigma$.

A homomorphism $h : \mathbb{A} \to \mathbb{B}$ is also an *isomorphism* when h is 1:1 and onto and the inverse function $h^{-1} : \mathbb{B} \to \mathbb{A}$ is a homomorphism.

2.10 Minimal Models

Minimal models, also called *universal* models, are models for a theory with the special property that there exists a homomorphism from the minimal model to any other model satisfied by the theory. Minimal models have no unnecessary entities or relations and thus display the least amount of constraint necessary to satisfy the theory for which they are minimal.

More than one minimal model may exist for a given theory, and not every theory must have a minimal model. **give examples**.

A set of models M is said to be *jointly minimal* for a set of formulæ Σ when every model μ such that $\mu \models \Sigma$ has a homomorphism from a model $m \in M$ to μ.

[1] geometric formulæ are implications of positive-existential formulæ

3 The Chase

talk about chase as nondeterministic algorithm or deterministic implementation algorithm? or both...

The *chase* is a function that, when given a gemoetric theory, will generate a set of jointly minimal models for that theory. More specifically, if U is the set of all models obtained from an execution of the chase over a geometric theory T, for any model $\mathbb{M}$ such that $\mathbb{M} \models T$, there is a homomorphism from some $u \in U$ to $\mathbb{M}$.

There are three types of runs of the chase:

- a non-empty result in finite time
- an empty result in finite time
- an infinite run, with possible return dependent on implementation

Recall that geometric formulæ are of the form

$$\forall\ (free(F_L) \cup free(F_R)) : F_L \rightarrow F_R$$

where $free$ is the function that returns the set of all free variables for a given formula and all F are first-order logic formulæ in positive existential form. Also recall that a geometric formula's implication is implicitly universally quantified over all free variables.

Geometric logic formulæ are used by the chase because they have the useful property where adding any relations or domain members to a model that satisfies a geometric logic formula will never cause the formula to no longer be satisified. This is particularly helpful when trying to create a model that satisifies all formulæ in a geometric theory.

3.1 Algorithm

3.2 Examples

Define Σ as the following geometric theory

$$\begin{aligned}
\top &\rightarrow \exists\ y, z : R[y, z] && (1)\\
R[x, w] &\rightarrow (\exists\ y : Q[x, y]) \vee (\exists\ z : P[x, z]) && (2)\\
Q[u, v] &\rightarrow (\exists\ z : R[u, z]) \vee (\exists\ z : R[z, w]) && (3)\\
P[u, v] &\rightarrow \bot && (4)
\end{aligned}$$

The following three chase runs show the different types of results depending on which disjunct the algorithm attempts to satisfy when a disjunction is encountered

TODO: label these

$$\begin{array}{lllll} \emptyset & \mapsto & \{ & a,b & \mid & R[a,b] & \} \\ & \mapsto & \{ & a,b,c & \mid & R[a,b],Q[a,c] & \} \end{array}$$

$$\begin{array}{lllll} \emptyset & \mapsto & \{ & a,b & \mid & R[a,b] & \} \\ & \mapsto & \{ & a,b,c & \mid & R[a,b],P[a,c] & \} \\ & \mapsto & \{ & a,b,c & \mid & R[a,b],P[a,c],\bot & \} \end{array}$$

$$\begin{array}{lllll} \emptyset & \mapsto & \{ & a,b & \mid & R[a,b] & \} \\ & \mapsto & \{ & a,b,c & \mid & R[a,b],Q[a,c] & \} \\ & \mapsto & \{ & a,b,c,d & \mid & R[a,b],Q[a,c],R[d,c] & \} \\ & \mapsto & \{ & a,b,c,d,e & \mid & R[a,b],Q[a,c],R[d,c],Q[d,e] & \} \\ & \mapsto & \{ & a,b,c,d,e,f & \mid & R[a,b],Q[a,c],R[d,c],Q[d,e],R[f,e] & \} \\ & \mapsto & \{ & a,b,c,d,e,f,\dots & \mid & R[a,b],Q[a,c],R[d,c],Q[d,e],R[f,e],\dots & \} \\ & \mapsto & \dots \end{array}$$

4 An Extended Application: Cryptographic Protocol Analysis

5 Haskell Chase Implementation

5.1 Future Considerations

A Table of Syntax

syntax	definition
f^{-1}	the inverse function of f
$R[a_0, a_1, a_2]$	a *relation* of: relation symbol R, arity 3, and tuple $\langle a_0, a_1, a_2 \rangle$
$\top$	a *tautological formula*; one that will always hold
$\bot$	a *contradictory formula*; one that will never hold
$\neg\alpha$	α does not hold
$\alpha \wedge \beta$	both α and β hold
$\alpha \vee \beta$	either α or β hold
$\alpha \rightarrow \beta$	either α does not hold or β holds
$\forall x : \alpha$	for each element of the domain as x, α holds
$\forall \vec{x} : \alpha$	for each $x_i \in x$, $\forall x_i : \alpha$ holds
$\exists x : \alpha$	for each at least one element of the domain as x, α holds
$\exists \vec{x} : \alpha$	for each $x_i \in x$, $\exists x_i : \alpha$ holds
$l[x \mapsto y]$	the environment l with variable x mapped to domain member y
$\mathbb{M} \models_l \sigma$	$\mathbb{M}$ *is a model of* σ under environment l
$\mathbb{M} \models \sigma$	$\mathbb{M} \models_l \sigma$ given any environment l
$\mathbb{M} \models_l \Sigma$	for each $\sigma \in \Sigma$, $\mathbb{M} \models_l \sigma$
$\mathbb{M} \models \Sigma$	for each $\sigma \in \Sigma$, $\mathbb{M} \models \sigma$
$\Sigma \models \sigma$	Σ *entails* σ
$\mathbb{M} \preceq \mathbb{N}$	there exists a *homomorphism* $h : \lvert\mathbb{M}\rvert \rightarrow \lvert\mathbb{N}\rvert$

B Chase code

TODO: make the long lines short so they fit

```
module Chase where
import Parser
import Helpers
import Debug.Trace
import Data.List

chaseVerify :: [Formula] -> [Formula]
-- verifies that each formula is in positive existential form and performs some
-- normilization on implied/constant implications
chaseVerify formulae =
   let isNotPEF = not.isPEF in
   map (\f -> case f of
      Implication a b ->
         if isNotPEF a || isNotPEF b then error ("implication must be in positive existential form:
         else f
      _ ->
         if isNotPEF f then error ("formula must be in positive existential form: " ++ showFormula
         else (Implication Tautology f)
   ) formulae

chase :: [Formula] -> [Model]
-- runs the chase algorithm on a given theory and returns a list of models that
-- satisfy it
chase formulae = chase' (chaseVerify formulae) ([],[(mkModel [] [])])

chase' :: [Formula] -> ([Model],[Model]) -> [Model]
-- used by the chase function to hide the model identity argument
chase' formulae (done,[]) = done
chase' formulae (done,pending) =
   let self = chase' formulae in
   let (p:ending) = pending in
   trace ("running chase on " ++ show (done,pending)) $
   if all (\f -> holds p (UniversalQuantifier (freeVariables f) f)) formulae then
      trace ("  all formulae in theory hold for model " ++ showModel p) $
      trace ("  moving model into done list") $
      self (union done [p],ending)
   else
      let possiblySatisfiedModels = attemptToSatisfyFirstFailure p formulae in
      trace ("  at least one formula does not hold for model " ++ showModel p) $
      trace ("  unioning " ++ show ending ++ " with [" ++ intercalate ", " (map showModel possiblyS
      self (done, union ending possiblySatisfiedModels)

attemptToSatisfyFirstFailure :: Model -> [Formula] -> [Model]
-- checks if each formula holds, sequentially, until one does not, then tries
-- to satisfy that formula
attemptToSatisfyFirstFailure model (f:ormulae) =
   let self = attemptToSatisfyFirstFailure model in
   if holds model (UniversalQuantifier (freeVariables f) f) then self ormulae
   else attemptToSatisfy model f

attemptToSatisfy :: Model -> Formula -> [Model]
-- returns a model that is altered so that the given formula will hold
attemptToSatisfy model formula =
   let f' = UniversalQuantifier (freeVariables formula) formula in
   trace ("  attempting to satisfy (" ++ showFormula formula ++ ")") $
   attemptToSatisfy' model [] f'

attemptToSatisfy' :: Model -> Environment -> Formula -> [Model]
-- hides the environment identity in the `attemptToSatisfy` function arguments
attemptToSatisfy' model env formula =
```

```
    let (domain,relations) = model in
    let domainSize = length domain in
    let self = attemptToSatisfy' model in
    -- trace ("  attempting to satisfy (" ++ showFormula formula ++ ") with env " ++ show env) $
    case formula of
        Tautology -> [model]
        Contradiction -> []
        Or a b -> union (self env a) (self env b)
        And a b -> concatMap (\m -> attemptToSatisfy' m env b) (self env a)
        Implication a b -> if holds' model env a then self env b else []
        Atomic predicate vars ->
            let newRelation = mkRelation predicate (length vars) [genNewRelationArgs env vars (fromInt
            let newModel = mkModel (mkDomain domainSize) (mergeRelation newRelation relations) in
            trace ("␣␣␣␣adding␣new␣relation:␣" ++ show newRelation) $
            [newModel]
        ExistentialQuantifier [] f -> self env f
        ExistentialQuantifier (v:vs) f ->
            let f' = ExistentialQuantifier vs f in
            let nextDomainElement = fromIntegral $ (length domain) + 1 in
            if any (\v' -> holds' model (hashSet env v v') f') domain then
                trace ("␣␣␣␣" ++ showFormula formula ++ "␣already␣holds") $
                [model]
            else
                trace ("␣␣␣␣adding␣new␣domain␣element␣" ++ show nextDomainElement ++ "␣for␣variable␣" +
                attemptToSatisfy' (mkDomain nextDomainElement,relations) (hashSet env v nextDomainEleme
        UniversalQuantifier [] f -> self env f
        UniversalQuantifier (v:vs) f ->
            let f' = UniversalQuantifier vs f in
            concatMap (\v' -> self (hashSet env v v') f') domain
        _ -> error ("formula␣not␣in␣positive␣existential␣form:␣" ++ showFormula formula)

genNewRelationArgs :: Environment -> [Variable] -> DomainElement -> [DomainElement]
-- for each Variable in the given list of Variables, retrieves the value
-- assigned to it in the given environment, or the next domain element if it
-- does not exist
genNewRelationArgs env [] domainSize = []
genNewRelationArgs env (v:ars) domainSize =
    let self = genNewRelationArgs env in
    case lookup v env of
        Just v' -> v' : (self ars domainSize)
        _ -> (domainSize+1) : (self ars (domainSize+1))
```

References

[1] A Cottrell, *Word Processors: Stupid and Inefficient*, `www.ecn.wfu.edu/~cottrell/wp.html`