2010-04-13

Margrave: An Improved Analyzer for Access-Control and Configuration Policies

Timothy Nelson
Worcester Polytechnic Institute

Margrave: An Improved Analyzer for Access-Control and Configuration Policies

by

Tim Nelson

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

April 2010

APPROVED:

Professor Kathi Fisler and **Professor Daniel Dougherty**, Thesis Advisors

---

Professor Craig Wills, Reader

Professor Michael Gennert, Head of Department

Abstract

As our society grows more dependent on digital systems, policies that regulate access to electronic resources are becoming more common. However, such policies are notoriously difficult to configure properly, even for trained professionals. An incorrectly written access-control policy can result in inconvenience, financial damage, or even physical danger. The difficulty is more pronounced when multiple types of policy interact with each other, such as in routers on a network.

This thesis presents a policy-analysis tool called Margrave. Given a query about a set of policies, Margrave returns a complete collection of scenarios that satisfy the query. Since the query language allows multiple policies to be compared, Margrave can be used to obtain an exhaustive list of the consequences of a seemingly innocent policy change. This feature gives policy authors the benefits of formal analysis without requiring that they state any formal properties about their policies.

Our query language is equivalent to order-sorted first-order logic (OSL). Therefore our scenario-finding approach is, in general, only complete up to a user-provided bound on scenario size. To mitigate this limitation, we identify a class of OSL that we call Order-Sorted Effectively Propositional Logic (OS-EPL). We give a linear-time algorithm for testing membership in OS-EPL. Sentences in this class have the Finite Model Property, and thus Margrave's results on such queries are complete without user intervention.

Acknowledgements

I could start this page by stating, simply:

I did it. Me! Me! All Me! Aren't I great?

But that would be a big, fat, smelly lie, and I'm *glad* it's not true. At least half the fun of this thesis has been in working with a few groovy people. This document would not exist without them.

My advisors, Profs. Kathi Fisler, Dan Dougherty, and Shriram Krishnamurthi, for introducing me to this field and for their encouragement.

My reader, Prof. Craig Wills, who has been supportive since my first day at WPI (and who has exercised saintly patience while waiting for this document!)

Danny, Theo, Yu, Guillaume, Ken, Chris, Paul, Roman and the rest of the ALAS lab and alumni. Working next to you all has been a pleasure.

Chris Barratt, my collaborator.

My wife Emily, who has dealt kindly with my Research Jitters (and who proof-read parts of this document.)

My family: Lisa, Erika, and Cary Nelson; John and Linda Gibbons. My dear friends Stef and Michelle, and all others whose names would not fit on this page.

This research has been supported by the National Science Foundation and graduate fellowships from WPI and the Carl and Inez Weidenmiller endowed fellowship. I am grateful for their generous support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the Internet has become more prevalent in society, so the typical person has needed to think more about issues of access control. Control is no longer a matter of buying a strong lock, but one of creating a strong policy. Furthermore, a policy must not be made excessively strong or legitimate agents may be unable to access needed resources. This process is often difficult. Many of us have asked questions such as: "Who has access to my home address in Facebook?", "Can anyone but my doctor read my medical history?", or "Why can't I see my spouse's Gmail calendar?". An incorrectly written access-control policy can result in inconvenience, financial damage, or even physical danger [Dum]. To complicate matters, policies often need to be modified. A new network host, a new (or terminated) employee, a new corporate doctrine – all these and more may induce changes to a working policy. Those changes must be implemented quickly and with confidence.

Oppenheimer, Ganapathi, and Patterson [OGP03] have shown that operator error is the major source of failure in online services, and that errors in configuration are the majority of such errors. So how does an administrator know that a policy (or policy change) reflects their intent? Without testing, even a technical expert may

inadvertently render their resources vulnerable [Woo04]. One option is to simply look for problems through testing. This option is problematic for several reasons: First, such testing is costly to perform correctly. Second, test environments are not available in many situations, which forces a user test a policy when it is already live. Third, vulnerability to certain attacks may not be possible to test on a live policy. Finally, testing is not a guarantee of proper functionality: A policy may pass a battery of tests, only to fail in a case that had not been considered by the tester.

A better option is to precisely state a set of properties that must hold about the policy, and to use formal methods to verify those properties. This provides a guarantee of proper functionality, something that even the most exhaustive suite of test cases lacks. In order to do this, one must first state what the important properties *are* for the policy; this is often difficult for users without training in logic and verification. A more subtle concern is this: how does a policy author know that they have stated *all* the important properties? A policy change could cause an unexpected error, which then becomes a new "important" property to test. This motivates a different kind of static analysis: examining the original and modified policies and generating a list of all situations in which they disagree. This is called "differencing" or "change-impact analysis". Such an analysis guarantees proper functionality since a policy author is likely to be able to recognize whether their *expectations* are met, even if they do not possess a formal *specification* [FKMT05]. Since the list of differences that change-impact analysis presents is exhaustive, the author knows that if those differences are as expected, there are no hidden cases lurking in the wings.

Change-impact analysis of policies was first proposed by Fisler *et al.* [FKMT05] but supported only a small subset of policies using propositional logic. Many real-world policies use richer language or are effectively the composition of many smaller

policies. The goal of this thesis is to extend verification and change-impact analysis to policies that involve a non-trivial notion of *environment*. Since firewall policies are known to be problematic for system administrators to write and maintain, we will pay special attention to that problem domain.

1.1 Summary of contributions

Our work builds off of Fisler *et al.* [FKMT05]. We allow policies and queries to involve predicates and limited quantification, yet show that they can still be soundly and completely addressed using propositional methods.

Furthermore, we do not limit ourselves to ordinary access-control: we provide a general framework that supports a wide variety of policy types and consider the interactions that can occur between policies in settings such as multi-router networks (Sections 3.2, 3.4).

The contributions in this thesis are threefold:

- We present a new tool for verification and change-impact analysis of policies. Since our work is in essence an expansion of Fisler *et al.* [FKMT05], we call our tool Margrave. The new tool supports far richer notions of policy and query than before, which has required a completely new implementation and a new theoretical approach.

- We identify a syntactically-determined class of sentences for which the Finite Model Property holds, rendering model-finding complete. We also give efficient algorithms for putting this theory into practice.

- We evaluate our work on a real-world enterprise firewall policy.

1.2 Thesis Roadmap

To set the stage for this thesis, we start with a motivating example in Chapter 2, then discuss the design and implementation of the Margrave tool in Chapter 3. We detail the necessary foundational work to support the tool in Chapter 4, including some theory-driven performance optimizations at the end of the chapter. We evaluate this thesis in Chapter 5, then summarize the related work in Chapter 6 before closing with perpective and future work in Chapter 7.

Chapter 2

A Motivating Example

2.1 Firewalls with NAT

One of the main goals of the new Margrave project is to move beyond simple access control. Networks are a major problem domain, and network tools need to reason about more than just isolated access-control policies. A firewall possesses access-control lists (ACLs) for each of its interfaces along with separate rule-sets for routing and network-address translation (NAT). Understanding how a firewall will act on a packet requires knowledge of which interface the packet is arriving on. Moreover, if we want to model networks in general, we need to consider multiple firewalls and any modifications they make to a packet in transit via NAT.

The following example is helpful in understanding why NAT adds an extra challenge to modeling firewall behavior. Suppose a small business has a group of employees, a group of contractors, a manager, a web server, and a mail server. Two firewalls isolate the servers in a demilitarized zone (DMZ). Figure 2.1 shows this toplogy in detail.

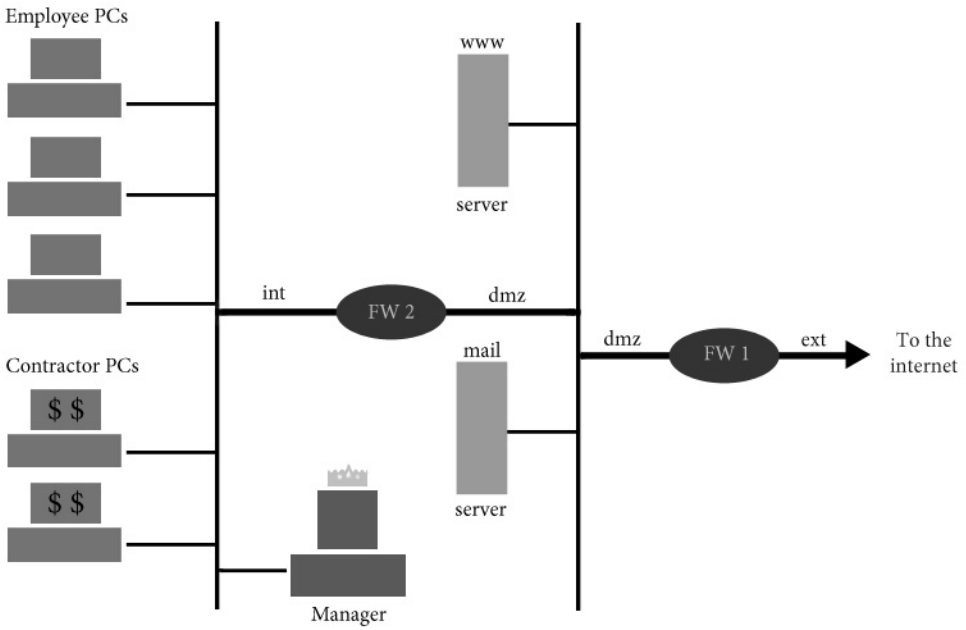


Figure 2.1: Example topology

The policies for the two example routers follow:

Inside Firewall's ACL:

```
Rule 1)   DENY if: interface=fw_inside_dmz
Rule 2)   DENY if: interface=fw_inside_internal, ipdest=mailserver,
              portdest=25, protocol=TCP, ipsrc in contractorPCs
Rule 3)   ACCEPT if: interface=fw_inside_internal, ipdest=mailserver,
              portdest=25, protocol=tcp
Rule 4)   ACCEPT if: interface=fw_inside_internal, portdest=80,
              protocol=tcp
Rule 5)   otherwise, DENY
```

This access-control list applies to traffic arriving at the inside firewall. It forbids any connections that originate from outside the protected network, while allowing all outgoing web traffic and permitting non-contractor PCs to access the company

mailserver.

Inside Firewall's NAT:

```
Rule 1)   Change ipsrc to fw_inside_static_outgoing if:
                interface=fw_inside_internal
```

This NAT policy stipulates that packets arriving at the internal interface of the inside firewall will be subjected to static NAT, changing their source IP address field.

Outside Firewall's ACL

```
Rule 1)   DENY if: interface=fw_outside_dmz, ipdest in blacklisted_ips
Rule 2)   DENY if: interface=fw_outside_external, ipsrc in blacklisted_ips
Rule 3)   DENY if: interface=fw_outside_dmz, portdest=23
Rule 4)   ACCEPT if: interface=fw_outside_external, ipdest=mailserver,
              portdest=25, protocol=tcp
Rule 5)   ACCEPT if: interface=fw_outside_external, ipdest=webserver,
              portdest=80, protocol=tcp
Rule 6)   ACCEPT if: interface=fw_outside_dmz, ipdest=any outside,
                     portdest=80, protocol=tcp, ipsrc=managerPC
Rule 7)   otherwise, DENY
```

This access-control list applies to traffic arriving at the outside firewall. It forbids any connections from blacklisted IPs as well as any outgoing telnet traffic. Outside hosts are allowed to use the mail and web servers on the appropriate ports, and only the manager's PC is allowed to surf the web.

Late one night, the company's system administrator gets a phone call from the manager. The manager can read e-mail, but not browse the web. The sysadmin has to fix the error quickly, but also without causing any new problems. This situation is exactly where Margrave can be most useful. The following Margrave query asks "When can a connection from the manager's PC be denied if it's to

port 80 somewhere outside our network?". As expected, the query takes NAT into account; see Section 3.4. We have edited the query content for clarity; Section 3.3 contains more information on the full query language.

```
EXPLORE port80(portdest) AND
        outsideIPs(ipdest) AND
        TCP(protocol) AND
        managerPC(ipsrc) AND
        fw_inside_internal(arrival1) AND
        fw_outside_dmz(arrival2)
        AND

        ( fw_inside_acl:drop(ipsrc, ipdest, portsrc, portdest, arrival1)
          OR
          ( fw_inside_nat:translate(ipsrc, intermip, portsrc, portdest, arrival1)
            AND
            fw_outside_acl:drop(intermip, ipdest, portsrc, portdest, arrival2)))
```

Margrave gives the following results [1]:

```
$arrival1 = fw_inside_internal
$arrival2 = fw_outside_dmz
$ipsrc = managerPC
$ipdest in outsideIPs
$portsrc = any
$portdest = port80
$protocol = tcp
```

This result confirms that the manager's workstation is forbidden to access the web. That isn't helpful by itself, but by activating output of applicable rules and re-running the query, he can learn:

```
fw_inside_acl's rule 4 accepts
  ($arrival1, $ipsrc, $ipdest, $portsrc, $portdest, $protocol).

fw_inside_nat translates
  ($arrival1, $ipsrc, $ipdest, $portsrc, $portdest, $protocol)
   into
  ($arrival1, $intermip, $ipdest, $portsrc, $portdest, $protocol).
```

[1] The "any" in the results represents several solutions that we have combined for brevity.

```
fw_outside_acl's rule 7 denies
  ($arrival1, $intermip, $ipdest, $portsrc, $portdest, $protocol)
```

This extra information helps the sysadmin infer *why* the problem happens: the outside firewall is dropping the packet since the inside firewall changed it as it passed; Rule 6 will not apply because the source IP is no longer *managerPC*.

Suppose that the sysadmin makes the "obvious" fix, changing the outside firewall's Rule 6 to accept packets coming from *fw_inside_static_outgoing*. Before updating the production system with the new policy he issues a change-impact query in Margrave, asking for an exhaustive list of all packets whose disposition changes. The query is shorter than it would normally have to be, because we are interested only in packets traveling outward, and only one ACL changes:

```
EXPLORE port80(portdest) AND
        outsideIPs(ipdest) AND
        TCP(protocol) AND
        managerPC(ipsrc) AND
        fw_inside_internal(arrival1) AND
        fw_outside_dmz(arrival2)
        AND

        fw_inside_acl:accept(ipsrc, ipdest, portsrc, portdest, arrival1) AND
        fw_inside_nat:translate(ipsrc, intermip, portsrc, portdest, arrival1)
        AND

        ( ( fw_outside_new_acl:accept(ipsrc, intermip, portsrc, portdest, arrival2)
            AND
            fw_outside_acl:drop(intermip, ipdest, portsrc, portdest, arrival2))
          OR
          ( fw_outside_new_acl:drop(ipsrc, intermip, portsrc, portdest, arrival2)
            AND
            fw_outside_acl:accept(intermip, ipdest, portsrc, portdest, arrival2)))
```

The results of this query will expose any unforseen consequences of the fix:

```
$arrival1 = fw_inside_internal
$arrival2 = fw_outside_dmz
```

```
$ipsrc = **any**
$ipdest in outsideIPs
$portsrc = any
$portdest = port80 (www)
$protocol = tcp
```

This confirms that the manager can now access the web, but so can everyone else! The original property being tested ("Can the manager access the outside web?") is itself incorrect; it doesn't reflect the sysadmin's intention. The actual property of interest is "Can the manager, *and nobody else* access the outside web?".

Of course, the problem is that there is no way for the outside firewall to tell who sent a packet once the inside firewall has altered the packet's header. One way to resolve this is to have the inside firewall filter web traffic while the outside firewall allows all web traffic from *fw_inside_static_outgoing*. After making this change, the sysadmin first tests the new property ("Can the manager, and nobody else..."), then performs a second change-impact analysis, seeking unintended side effects. This time, there are none.

2.2 Summary

The iterative process in the above example illustrates a major use case for change-impact analysis. Changing a policy can often have unforseen side effects. Production systems are sensitive to such side-effects. Our tool is designed to guide policy authors as they make changes, preventing them from compromising their system in the process.

Chapter 3

Margrave Internals

3.1 Architecture

Margrave consists of a Scheme front-end and a query engine written in Java. We use the Kodkod [TJ07] model-finder as our back-end along with the SAT4j SAT Solver library[1]. This relationship is sketched in 3.1.

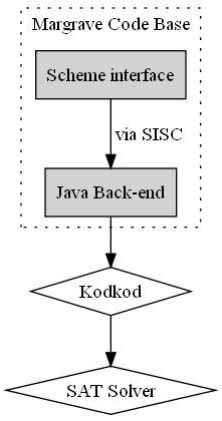


Figure 3.1: Margrave Architecture

[1]Available at http://www.sat4j.org

Having a Scheme REPL[2] as a front-end lets users interact naturally with Margrave via a command-line interface. Users can load policies, define and run queries, set environment variables and process query results all from the same command-line environment. They can also write query scripts offline using Scheme and load them in Margrave. We use a Scheme implementation called SISC (Second Interpreter of Scheme Code) [Mil02] for this because it is itself implemented in Java, giving us a convenient foreign-function interface to our engine and to Kodkod. SISC provides the entire R5RS Scheme language standard.

The REPL model encourages an iterative query process. A user may start by asking "What requests will policy A and policy B disagree on?", then refine their results, looking for unexpected consequences: "Do any of those requests involve students?", "Do any of those requests involve students reading final grades?", etc.

Margrave can present the results of a query in different ways:

- *Is the query satisfiable?*: Returns either a true or false.
- *Print all solutions*: Gives a stream of first order-models that satisfy the query.
- *Print the first solution*: Prints only the first model found that satisfies the query.

When displaying a model, Margrave prints its size and a visualization of what each relation contains. Relations are either EDBs (Extensional Data Bases, representing ground facts) or IDBs (Intensional Data Bases, representing policy-specific facts such as decision or rule applicability). Constants (i.e., existential variables in the query prefix) are shown by name, along with what value they take in the model.

[2]An acronym for Read-Eval-Print-Loop. A REPL is an interactive programming environment and is often seen in Lisp.

Figure 3.2: Left: A simple model. Right: A model with IDB Output

```
*** SOLUTION: Size = 3.
$subject: reviewer author
$action: readpaper
$resource: paper
conflicted =
  { [$subject, $resource] }
assigned = {}
```

```
*** SOLUTION: Size = 3.
$subject: reviewer author
$action: readpaper
$resource: paper
conflicted =
  { [$subject, $resource] }
assigned = {}
policy1:rule3_applies =
  { [$subject, $action, $resource] }
policy1:deny =
  { [$subject, $action, $resource] }
```

Figure 3.1 (left) shows what a simple model of a paper review system request looks like when printed.

If a model is unexpected, the user can discover which policies and rules are responsible for the change by enabling a feature called IDB output. This feature causes model output to include IDBs as well as EDBs, explaining *why*, rather than just *what*. Figure 3.1 (right) shows a simple model with IDB output included. Since IDB output adds additional relations to the query's language, it can add a significant overhead to execution time, depending on the arity of the IDBs being shown.

3.2 Margrave Policy Language

Margrave is applicable to policies over arbitrary problem domains, even to the combination of different kinds of policies. Each domain has its own set of concepts that Margrave must represent. For instance, a firewall policy cares about IP addresses, ports, and protocols. A conference manager's policy involves authors, papers, and conflicts of interest. These concepts define the logical language in which a policy is written and over which a Margrave query may be executed. It is therefore useful to categorize policies by their *vocabulary*, and so we have separated policies from their

vocabularies in our language.

3.2.1 Vocabulary

A Margrave vocabulary defines the ontology of a policy: What sorts are available (subject, action, and resource? Perhaps IP addresses and ports, instead?), non-sort predicates, and contraints are all defined in the vocabulary. Every policy uses exactly one vocabulary. The sorts and predicates in the vocabulary are called the EDB symbols.

The vocabulary also defines what shape a policy request takes, in the form of variable declarations. For instance, if the requests are TCP/IP packets, the vocabulary would declare names for each of the packet header fields.

Finally the vocabulary lists what decisions may be rendered. While "Permit" and "Deny" are standard, Margrave allows a policy to be more fine-grained, letting the user make a distinction between "Deny" and "Drop", or including a "Deny and log" decision. If a vocabulary is meant to model the interaction of different kinds of policies, it may declare decisions from both. For instance, a vocabulary modelling firewall ACLs and NAT in tandem may declare "Permit", "Deny", and "Translate" decisions.

Margrave allows the following constraints to be imposed:

- *Disjointness*: Two sorts A and B must be disjoint.
- *at-most-one (also known as LONE)*: Sort A may contain at most one element.
- *nonempty (also known as SOME)*: Sort A must contain at least one element.
- *singleton*: Sort A contains exactly one element.
- *abstract*: Sort A is equal to the union of its subsorts (if it has subsorts).
- *partial function*: Predicate $F \subseteq (A_1 \times ... \times A_n)$ is a partial function from $(A_1 \times ... \times A_{n-1})$ to A_n.

- *partial function*: Predicate $F \subseteq (A_1 \times ... \times A_n)$ is a total function from $(A_1 \times ... \times A_{n-1})$ to A_n.

Figure 3.3 shows a sample vocabulary. It is a vocabulary for phone company policies, which take source and destination phone numbers and decide whether to refuse the call, charge toll, or make the call toll-free. The function GetExchange represents the function mapping numbers to their exchange.

```
(PolicyVocab PhonePolicy
             (Types
              (Number : InService OutOfService)
              (Exchange : AnExchange ))
             (Decisions
              TollFree
              Toll
              Refuse)
             (Predicates
              (GetExchange : Number Exchange))

             (ReqVariables (src : Number)
                           (dest : Number))
             (OthVariables (e1 : Exchange) (e2 : Exchange) (e3 : Exchange))
             (Constraints

              ; There is a unique exchange for each number.
              (total-function GetExchange)

              ; A number is either in or out of service, never both.
              (disjoint-all Number)

              (abstract Exchange)
              (abstract Number)

              ; Weed out vacuous solutions
              (nonempty Number)
              (nonempty Exchange)))
```

Figure 3.3: Sample Vocabulary

3.2.2 Policy (Leaf and Set)

A Policy is either a Leaf or a Policy Set. A policy Leaf is a standalone access-control policy: it consists of a set of rules and a rule-precedence[3] setting. A policy Set consists of a policy-precedence setting and a set of child policies. Policy sets allow Margrave to support hierarchical policies of the kind used extensively in XACML.

Margrave accepts the following precedence settings: *first-applicable*, in which the first rule (or policy) in the list that applies takes precedence, and *overrides*: Given a total ordering on the set of decisions, higher-priority decisions take precedence. A policy *rule* is a set of EDB literals and a decision. The literals may be either positive or negative, may involve both the request variables declared in the vocabulary as well as rule-scope existential variables. For instance, a rule may say: "Permit if the subject is a customer and there exists some manager who has approved the customer's application."

From the given vocabulary, policies, and rules, Margrave constructs first-order formulas representing when each rule applies and when each policy decision is rendered. For a sample policy file, see Figure 3.4. This sample uses the vocabulary in Figure 3.3. Senseless requests are refused while intra-exchange calls are toll-free. Since the company wants to avoid giving free calls away, the Refuse decision overrides TollCall, which in turn overrides TollFree.

3.2.3 Other Supported Languages

Beyond our own intermediate language, we have implemented parsers for several standard access-control languages that are in wide use. We support:

- XACML 1.0, 1.1, and 2.0

[3]Called combining algorithms in XACML.

(**Policy** *Phone1* **uses** *PhonePolicy*
(**Target**)
(**Rules**
(*TollFreeCall* = (*TollFree src dest*) :-
(*GetExchange src e1*) (*GetExchange dest e1*))
(*TollCall* = (*Toll src dest*) :-
(*GetExchange src e2*) (*GetExchange dest e3*) (*!= e2 e3*))
(*RefuseCall1* = (*Refuse src dest*) :- (*OutOfService src*))
(*RefuseCall2* = (*Refuse src dest*) :- (*OutOfService dest*))
(*RefuseCall3* = (*Refuse src dest*) :- (*= src dest*)))
(**RComb** *O Refuse Toll TollFree*)
(**PComb** *O Refuse Toll TollFree*)
(**Children**))

Figure 3.4: Sample Policy

- Amazon SQS policy language
- Cisco IOS

XACML The XACML [OAS05] interface uses Sun's XACML Implementation[4]. We support the TARGET element and most CONDITION elements. Complex functional CONDITION elements are supported by treating each application as describing a predicate name. For instance the following function application, representing that the current time must be at least 9 am:

```
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
    <EnvironmentAttributeSelector
       DataType="http://www.w3.org/2001/XMLSchema#time"
       AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
  </Apply>
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
</Apply>
```

[4]Available at: http://sunxacml.sourceforge.net/

becomes a single predicate over Environments. We do not consider the semantics of these condition predicates – for instance, given two predicates ($S.age < 18$) and ($S.age \geq 18$) we do not force them to be disjoint. We also do not consider XACML's bag semantics.

Amazon SQS Amazon Simple Queue Services is a distributed queue system for use by web applications. Recently, Amazon announced a basic access-control language for use with SQS queues[5].

The core language is strictly less expressive than XACML, and so it was not difficult to implement after our work on XACML. Amazon plans to expand other services to use the language soon; any expansions made to the language can be accounted for as needed.

IOS Cisco's (Internetwork Operating System) IOS is a configuration language widely used on Cisco routers and firewalls.

The IOS interface for Margrave supports standard and extended ACLs, static NAT, ACL-based and map-based dynamic NAT, static routing, and policy-based routing. The IOS parser for Margrave is due to Christopher Barratt, who was a graduate student at Brown University while this research was underway. We have collaborated with him on applying Margrave to IOS policies.

Since a Cisco IOS configuration file may contain more than a simple ACL, we also handle NAT and routing instructions. These components of the configuration are treated as separate Margrave policies, and can be used together in queries. For example, we may ask which packets will be routed by a firewall.

[5] `http://docs.amazonwebservices.com/AWSSimpleQueueService/2009-02-01/SQSDeveloperGuide/`

```
EXPLORE InboundACL:permit(ahostname, entry-interface, src-addr-in, src-addr-in,
  dest-addr-in, dest-addr-in, protocol, message, src-port-in, src-port-in,
  dest-port-in, dest-port-in, length, next-hop, exit-interface) AND

  OutsideNAT:translate(ahostname, entry-interface, src-addr-in, src-addr_,
  dest-addr-in, dest-addr_, protocol, message, src-port-in, src-port_,
  dest-port-in, dest-port_, length, next-hop, exit-interface) AND

(
  LocalSwitching:Forward(ahostname, entry-interface, src-addr_, src-addr_,
  dest-addr_, dest-addr_, protocol, message, src-port_, src-port_,
  dest-port_, dest-port_, length, next-hop, exit-interface)
  OR

  (
    LocalSwitching:Pass(ahostname, entry-interface, src-addr_, src-addr_,
    dest-addr_, dest-addr_, protocol, message, src-port_, src-port_,
    dest-port_, dest-port_, length, next-hop, exit-interface)
    AND
    (
      PolicyRoute:Forward(ahostname, entry-interface, src-addr_, src-addr_,
      dest-addr_, dest-addr_, protocol, message, src-port_, src-port_,
      dest-port_, dest-port_, length, next-hop, exit-interface)
      OR
      (
        PolicyRoute:Route(ahostname, entry-interface, src-addr_, src-addr_,
        dest-addr_, dest-addr_, protocol, message, src-port_, src-port_,
        dest-port_, dest-port_, length, next-hop, exit-interface)
        AND
        NetworkSwitching:Forward(ahostname, entry-interface, src-addr_,
        src-addr_, dest-addr_, dest-addr_, protocol, message,
        src-port_, src-port_, dest-port_, dest-port_, length,
        next-hop, exit-interface))
      OR
      (
        PolicyRoute:Pass(ahostname, entry-interface, src-addr_, src-addr_,
        dest-addr_, dest-addr_, protocol, message, src-port_,
        src-port_, dest-port_, dest-port_, length, next-hop,
        exit-interface)
        AND
        (
          StaticRoute:Forward(ahostname, entry-interface, src-addr_,
          src-addr_, dest-addr_, dest-addr_, protocol, message,
          src-port_, src-port_, dest-port_, dest-port_, length,
          next-hop, exit-interface)
```

```
          OR
          (
            StaticRoute:Route(ahostname, entry-interface, src-addr_,
            src-addr_, dest-addr_, dest-addr_, protocol, message,
            src-port_, src-port_, dest-port_, dest-port_, length,
            next-hop, exit-interface)
            AND
            NetworkSwitching:Forward(ahostname, entry-interface, src-addr_,
            src-addr_, dest-addr_, dest-addr_, protocol, message,
            src-port_, src-port_, dest-port_, dest-port_, length,
            next-hop, exit-interface))
          OR
          (
            StaticRoute:Pass(ahostname, entry-interface, src-addr_, src-addr_,
            dest-addr_, dest-addr_, protocol, message, src-port_,
            src-port_, dest-port_, dest-port_, length, next-hop,
            exit-interface)
            AND
            (
              DefaultPolicyRoute:Forward(ahostname, entry-interface,
              src-addr_, src-addr_, dest-addr_, dest-addr_,
              protocol, message, src-port_, src-port_, dest-port_,
              dest-port_, length, next-hop, exit-interface)
              OR
              (
                DefaultPolicyRoute:Route(ahostname, entry-interface,
                src-addr_, src-addr_, dest-addr_, dest-addr_,
                protocol, message, src-port_, src-port_, dest-port_,
                dest-port_, length, next-hop, exit-interface)
                AND
                NetworkSwitching:Forward(ahostname, entry-interface,
                src-addr_, src-addr_, dest-addr_, dest-addr_,
                protocol, message, src-port_, src-port_, dest-port_,
                dest-port_, length, next-hop, exit-interface)))))))))

AND InsideNAT:Translate(ahostname, entry-interface, src-addr_, src-addr-out,
  dest-addr_, dest-addr-out, protocol, message, src-port_,
  src-port-out, dest-port_, dest-port-out, length, next-hop,
  exit-interface)
AND OutboundACL:Permit(ahostname, entry-interface, src-addr-out, src-addr-out,
  dest-addr-out, dest-addr-out, protocol, message, src-port-out,
  src-port-out, dest-port-out, dest-port-out, length, next-hop,
  exit-interface)

PUBLISH entry-interface, src-addr-in, dest-addr-in, src-port-in, dest-port-in
```

We plan to provide "stock" queries such as this in the Margrave API, so that users do not need to enter them manually, but may use them as custom IDBs in their own queries.

Other Languages Interfaces for similar languages can be added via Margrave's Java API.

3.3 Margrave queries

Margrave allows users to write queries using order-sorted first-order logic. Queries may use arbitrary quantification and refer to EDBs as well as policy decisions and rules (IDBs) by name. The formal grammar is:

A query is one of:

- `(and QUERY_1 ... QUERY_n)`
- `(or QUERY_1 ... QUERY_n)`
- `(forsome varname sortname QUERY)`
- `(forall varname sortname QUERY)`
- `(not QUERY)`
- `(= var_1 var_2)`
- `(edbname var_1 ... var_k)`
- `(idbname var_1 ... var_k)`

We are also working on providing a user-friendly interface for purely existential queries; the examples above that use the "EXPLORE" syntax demonstrate this work in progress.

3.3.1 Query Handling

Once a query is submitted, the vocabulary provides constraint axioms that are then combined with the query. The result is a sentence in order-sorted first-order logic that exactly represents the query. To discover whether the query's conditions can be met, we must decide whether or not the sentence is satisfiable — an undecidable problem in general!

We have circumscribed a decidable class of order-sorted logic, OS-EPL, which is discussed in detail in Chapter 4. This class is decidable because given a sentence, we can produce bounds on the model sizes we need to check when searching for models. If such bounds exists, model-finding is a complete decision procedure for satisfiability.

We have incorporated this theory into Margrave. Given a query sentence, the tool first checks to see if the sentence falls into OS-EPL. If so, it searches for models up to the sufficient ceiling. If the sentence is not in OS-EPL, the user is alerted and must provide a bound, which is what other model-finding tools such as Alloy ([Jac06]) and Paradox ([CS03a]) require.

For sentences in OS-EPL, Margrave's results are *exhaustive*: any model of the query sentence will have a submodel no bigger than the established bound. This means that in spite of the fact that the tool will not list *all* models of the sentence (and indeed doing so would be bad: there might be infinitely many such models!) it will express all possible *root causes* that satisfy the query. This property is a consequence of our finite model theorem in Section 4.4.

Sample queries appear below in Figures 3.5 and 3.6. They both use the policy in Figure 3.4. Figure 3.5 asks what sorts of calls can be classified as Toll Free. Figure 3.6 is a change-impact query between the original policy and a new version. We provide an API function in Margrave that quickly produces a standard change-

impact query, but also allow users to write their own. The query in Figure 3.6 illustrates the second option.

```
   ; pPhone1 loaded beforehand
(query-policy pPhone1
(forsome ncaller Number
  (forsome nreceive Number
    (Phone1:TollFree ncaller nreceive))))
(pretty-print-results qry)
```

Figure 3.5: A query (1)

```
   ; pPhone1 and pPhone2 loaded beforehand
   ; suppose pPhone2 is a new version of pPhone1
(query-policy pPhone1
(forsome ncaller Number
  (forsome nreceive Number
    (or (and (pPhone1:TollFree ncaller nreceive)
             (not (pPhone2:TollFree ncaller nreceive)))
        (and (pPhone1:Toll ncaller nreceive)
             (not (pPhone2:Toll ncaller nreceive)))
        (and (pPhone1:Refuse ncaller nreceive)
             (not (pPhone2:Refuse ncaller nreceive)))
        (and (pPhone2:TollFree ncaller nreceive)
             (not (pPhone1:TollFree ncaller nreceive)))
        (and (pPhone2:Toll ncaller nreceive)
             (not (pPhone1:Toll ncaller nreceive)))
        (and (pPhone2:Refuse ncaller nreceive)
             (not (pPhone1:Refuse ncaller nreceive)))))))
```

Figure 3.6: A query (2)

It is possible to handle some special classes of queries differently in order to get a performance increase. We discuss a particular special class later in this section.

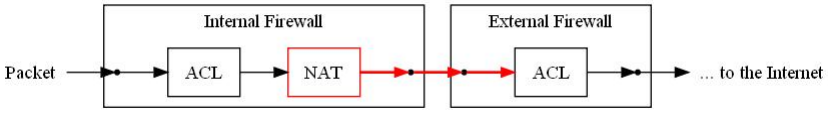
3.3.2 Vacuity

An interesting complication arises from allowing full-first order queries, yet trying to keep model sizes as small as possible. Suppose that a policy involves 10 individual IP addresses that we model as subsorts of IPAddress. Having a separate atom for each of those addresses might make query execution slow, so instead of declaring each of those sorts to have *exactly* one atom (via the "one" constraint) we instead say that they have *at most* one atom each (via the "lone" constraint). This causes no problems if the query is existential, but suppose it contains a universal quantifier over the IPAddress sort. Now there may be many models that satisfy the query only because they lack atoms in the important IP Addresses.

3.4 Using Margrave to model networks

We saw in section 2.1 that modeling networks or even single firewalls with NAT can be tricky. To recap, suppose a packet is sent and must traverse multiple firewalls en route (Figure 3.7). If the first firewall performs NAT on the packet, it changes the packet header. In our terminology, it has effectively *altered the request* mid-way through its evaluation!

Figure 3.7: NAT alters a request as it is evaluated.



In order to deal with this situation in our model, we take two steps. First, each firewall's ACLs and NAT policies are treated as separate Margrave policies. Second, queries involving NAT must consider changes to the original packet by adding extra

variables that represent the intermediate state of the (possibly) altered packet fields; this is exactly what our proverbial sysadmin did back in section 2.1.

We consider it a weakness in the tool that users must write these queries manually. Improving the query language and the interface for different problem domains is something we are actively working on; see section 7.2.

3.5 Performance Optimizations

Rule Precedence As one might expect, the internal representation of a formula in Margrave can greatly affect performance. An unsimplified formula will consume more memory and take more time to handle, but simplification is expensive. We try to express formulas as concisely as possible from the beginning. A good example of this is our treatment of the first-applicable rule precedence algorithm. A rule R applies under first-applicable only if rules appearing *before* R in the policy do not. Naively, one might construct a formula for Permit by taking the disjunction of the applicability formulas for all the Permit rules. This approach results in a formula whose size is quadratic in the number of rules. It is possible to create a logically equivalent, linear-sized formula. Algorithm 1 builds such formulas for each decision:

Algorithm 1: First Applicable

```
let Dec be vector of formulas, one for each decision ;
initialize each element of Dec to ⊥ ;
for each rule R (with decision D) in reverse order do
    D = R ∨ D ;
    for each decision D' ≠ R's decision do
        D' = D' ∧ ¬R ;
```

This approach is especially valuable when modelling firewall policies, which al-

ways combine rules via first-applicable and are often very large. We found that on a simulated 1000-rule firewall policy, this approach reduced formula size by 2 orders of magnitude.

Canonical Map Formulas in Margrave are stored internally as Kodkod abstract-syntax objects. Since Kodkod does not maintain a canonical map for its formulas, it may produce many isomorphic (but not equal) copies of the same formula.

We implemented our own canonical map using weak references. This greatly reduced the number of objects in memory; the actual savings depends on the amount of duplication within a formula, but we see a reduction of 3 to 4x on most of our large tests. More importantly having a canonical map allowed us to make proper use of caching when processing formula trees.

Tupling We implement the tupling optimization described in Section 4.7 for existental queries without functional constraints. Tupling can give a substantial boost in speed (Figures 3.1 and 3.2) that is primarily dependent on the size of the request vector, which is what one would expect. It also does better when the original formula's smallest model is large compared to bound generated by algorithm 2. This is because smaller model sizes are checked first [6].

Tupling can also reduce the number of solutions returned from Kodkod as well. By way of example, consider the following sentence: $\sigma \equiv \exists x\ \exists y\ \exists z\ P(x, y, z)$. Suppose we want to use model-finding to characterize all models of σ. Naively, this involves looking at each model of size up to 3 (since we know from Section 4.4 that each model of σ has a "small" submodel that also satisfies σ). In general, even with the help of Kodkod's symmetry-breaking techniques, there may be too many models

[6]Benchmarking trial results are the average of n runs per category, where $n = 100$ for the 100-rule trials and $n = 10$ for the 1000-rule trials.

to enumerate them all. For example, consider the tupled version of σ, $\exists z P_{1,2,3}(z)$ versus the original. Figure 3.3 gives the number of models for both formulas at each size up to 3. The single size 1 model of the tupled formula characterizes each of the millions of models of the original formula at size 3. Even if equality is explicitly considered (in this case, it need not be) there are still only 5 models of the tupled query.

Table 3.1: Without tupling: n rules per policy, k-ary request vector

n	k	smallest model	avg is-sat?
100	3	1	141ms
100	3	3	691ms
1000	14	6	7923ms
1000	14	10	18239ms
1000	14	14	32751ms

Table 3.2: With tupling: n rules per policy, k-ary request vector, only one solution when tupled

n	k	smallest model	avg is-sat?	avg portion of that used for Tupling
100	3	1	19ms	17ms
100	3	3	212ms	64ms
1000	14	6	1053ms	457ms
1000	14	10	1115ms	493ms
1000	14	14	1148ms	558ms

Table 3.3: Number of satisfying models

Size	Original (w/o s.b.)	Original (w/ s.b.)	Tupled σ'	Tupled σ' with equality
1	1	1	1	8
2	1024	512	-	-
3	millions	millions	-	-

Chapter 4

Foundations: Order-Sorted Logic

4.1 Introduction

The Schoenfinkel-Bernays-Ramsey class, or sometimes, "Effectively Propositional Logic" (EPL), comprises the set of first-order sentences of the form

$$\exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_m \; . \; \phi$$

where ϕ is quantifier-free and has no function symbols. The satisfiability problem for this class is decidable: Schoenfinkel and Bernays [BS28] and Ramsey [Ram30] showed that such a sentence has a model if and only if it has a model of size bounded by n plus the number of constants in ϕ.

When such a finite model property holds, satisfiability-testing can be reduced to exhaustive search. More important to applications is the fact that model-finders for EPL sentences can restrict their search to models whose elements are constants; similarly, instantiation-based theorem provers can restrict attention to instantiation by constants.

Model-finders and theorem-provers can benefit from the additional information

that a many-sorted framework can provide [Jer88, HRCS02, dMB08b, LS04]. More strikingly, the class of sentences supporting finite model theorems is richer in a many-sorted framework than in the one-sorted case [Harpt, FG03, ARS10]. In order to meet our goals for Margrave, we have made a systematic study of this latter phenomenon, in a very general *order-sorted* framework. We identify a broad class of sentences comprising *Order-Sorted Effectively Propositional Logic (OS-EPL)*, for which the Finite Model Property holds (Theorem 4), and present algorithms that allow us to treat OS-EPL both soundly and completely in Margrave.

We have also created a Web interface (`http://sortedtermcount.appspot.com`) for readers to experiment with the algorithms we describe here and use in Margrave. The tool accepts input in the notation used by Alloy [Jac06], a popular system for the analysis of system requirements and designs.

The following simple example gives the flavor of our results. Consider the class of unsorted sentences of the form

$$\forall y_1 \exists x \forall y_2 \; . \; \phi.$$

This prefix class has an undecidable satisfiability problem. But the following sorted version

$$\sigma \equiv \forall y_1^A \exists x^B \forall y_2^A \; . \; \phi \tag{4.1}$$

is better-behaved. Suppose that ϕ contains constants, say n_A constants of sort A and n_B of sort B, but no function symbols. Suppose in addition that sort A is a subsort of sort B. Under these conditions σ is in OS-EPL. Indeed we can show that if σ has any models at all then it has a model whose size at sort A is bounded by n_A and whose size at sort B is bounded by $(2n_A + n_B)$. So we have a finite model theorem and satisfiability for this class of formulas is decidable. On the other hand

if we were to require instead that B is a subsort of A, then the resulting sentence is not in OS-EPL; some such σ have only infinite models. These assertions are all consequences of our main theorem, Theorem 4 below.

Outline An instructive way to present the technical challenges and contributions of this work is to consider a standard approach to showing that a class of sentences has the finite-model property. Given a sentence σ in unsorted first-order logic we might reason as follows.

1. By Skolemization, there is a universal sentence σ_{sk} equi-satisfiable with σ;

2. Any potential model $\mathcal{M}$ for σ_{sk} has a *Skolem hull*, obtained by closing the interpretation of the constants by the interpretation of the functions [CK73]. This makes a submodel of $\mathcal{M}$ in which every element is named by a term in the language.

3. An easy theorem of unsorted logic is that the truth of universal sentences is preserved under submodel.

Thus, if the signature of σ_{sk} has only finitely many terms, we can conclude our finite model theorem. This last part of the argument succeeds in the one-sorted case only for formulas in the EPL class, but the starting point of this research is the observation that the many-sorted framework offers more opportunities for a finite Herbrand universe. However, the following facts complicate matters.

1. When empty sorts are allowed, the Skolem form of σ is not equi-satisfiable with σ (Section 4.3).

2. When sorts are not assumed to be disjoint—in particular, in the order-sorted setting—not every element in the Skolem hull of a model is named by a term. Indeed the Skolem hull of $\mathcal{M}$ can be infinite even when a finite submodel

of $\mathcal{M}$ *does* exist (Example 6). (This phenomenon also cause difficulties in giving a model-theoretic proof of Herbrand's Theorem for order-sorted logic: see Section 4.4.4).

3. When sort names are allowed to be used as predicates, as they are in many tools, preservation of universal sentences under submodel fails (Section 4.6).

Our development addresses each of these difficulties.

We view the identification of the OS-EPL class as a contribution to a taxonomy of decidability classes in order-sorted logic in the same spirit as for classical unsorted logic. In the presence of possibly-empty sorts, sentences do not always have equivalent prenex-normal forms, so we cannot attempt a decidability classification in terms of quantifier prefix as in [BGG97]. As Section 4.4 shows, our decidability criterion is based entirely on the signature of the Skolemization of the given formula. This signature can be viewed as a generalization of the idea of quantifier prefix, as it implicitly records the pattern of nesting between universal and existential quantification.

4.2 Preliminaries: Order-Sorted Predicate Logic

The definitions and results in this section are either directly from Goguen and Meseguer's work [GM92] or they are the obvious extensions to the setting in which relations as well as function are considered.

Notation We use $\langle\rangle$ for the empty sequence. If $(\mathcal{S}, \leq)$ is an ordering we extend $\leq$ to words in $\mathcal{S}^*$ and then to products, pointwise. The *connected components* of $(\mathcal{S}, \leq)$ are the equivalence classes for the equivalence relation generated by $\leq$.

Suppose $w = A_1 \dots A_n$ is a word in $\mathcal{S}^*$ and suppose that for each i we have defined a set X_{A_i}; then the notation X_w refers to $X_{A_1} \times \cdots \times X_{A_n}$.

Signatures An *order-sorted signature* is a triple $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$ where $(\mathcal{S}, \leq)$ is a finite poset of sorts and Σ is an indexed family of symbols, the vocabulary, comprising

- $\{\Sigma_w \mid w \in \mathcal{S}^*\}$, an $\mathcal{S}^*$-sorted family of *relation symbols,* and
- $\{\Sigma_{w,A} \mid w \in \mathcal{S}^* , A \in \mathcal{S}\}$, an $(\mathcal{S}^* \times \mathcal{S})$-sorted family of *function symbols*, satisfying the monotonicity condition

$$f \in \Sigma_{w_1,A_1} \cap \Sigma_{w_2,A_2} \quad \text{and} \quad w_1 \leq w_2 \quad \text{imply} \quad A_1 \leq A_2$$

When $R \in \Sigma_w$ we say that w is the *arity* of R. When $f \in \Sigma_{w,s}$ we say that w is the *arity* of f and A is the *result sort* of f.

A signature is *regular* if whenever $f \in \Sigma_{w_1,A_1}$ and $w_0 \leq w_1$ then there is least $(w, S) \in (\mathcal{S}^* \times \mathcal{S})$ such that $w_0 \leq w$ and $f \in \Sigma_{w,s}$. A signature $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$ is *coherent* if it is regular and $(\mathcal{S}, \leq)$ is locally filtered (*i.e.*, each pair of sorts in the same connected component has an upper bound).

If $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$ and $\mathcal{L}' = (\mathcal{S}, \leq, \Sigma')$ are such that for each w and A, $\Sigma_w \subseteq \Sigma'_w$ and $\Sigma_{w,A} \subseteq \Sigma'_{w,A}$ we say that $\mathcal{L}'$ is an *expansion* of $\mathcal{L}$, and that $\mathcal{L}$ is a *reduct* of $\mathcal{L}'$.

This is a very general notion of signature, allowing symbols to be overloaded, *i.e.*, declared in more than one sort. Following standard usage, a symbol $a \in \Sigma_{\langle\rangle,A}$ is referred to as a "constant" of sort A, and in concrete syntax we write simply a instead of $a()$. Note that the monotonicity assumption means that constants cannot be overloaded.

On the other hand, note that sort names are *not* eligible to be used as predicates, in contrast to certain treatments of many-sorted logic. There are very good reasons for this: see Section 4.6.

Example 1. *The following signature is filtered, but* not *regular.* $\mathcal{S} = \{A, B, C, D\}$, *with* $A \leq B \leq D$ *and* $A \leq C \leq D$. $\Sigma_{\langle\rangle,A} = a$, $\Sigma_{B,B} = f$, $\Sigma_{C,C} = f$. *Not regular since there is no least* (w, S) *with* $A \leq w$ *and* $f \in \Sigma_{w,S}$.

See Example 3, Example 4, and Theorem 1 below for motivation for the restriction to coherent signatures.

The restriction to finite vocabulary is reasonable given the fact that our main concern in this paper is finite model theorems. In our setting, when $\mathcal{S}$ is finite, local filtering is equivalent to the requirement that each component have a maximum element.

In general, we would like to specify that certain pairs of sorts are to be disjoint (*e.g.* Alloy, with "extends"). For simplicity we do not consider such a mechanism in our notion of signature, since we can get the same effect by explicitly writing disjointness sentences and conjoining hem with any sentences under consideration (as long as they are in the same connected component!). For future reference we note that such sentences involve no existential quantifiers.

Models Fix an OS signature $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$. An $\mathcal{L}$-model $\mathcal{M}$ comprises

- an $\mathcal{S}$-sorted family $\{\mathcal{M}_A \mid A \in S\}$ of sets, the *universe* of $\mathcal{M}$, such that

$$A \leq A' \text{ implies } \mathcal{M}_A \subseteq \mathcal{M}_{A'},$$

- for each $R \in \Sigma_w$ a relation $R^{\mathcal{M}_w} \subseteq \mathcal{M}_w$, such that

$$R \in \Sigma_{w_1} \cap \Sigma_{w_2} \quad \text{and} \quad w_1 \leq w_2 \quad \text{imply} \quad R^{\mathcal{M}_{w_1}} = R^{\mathcal{M}_{w_2}} \cap \mathcal{M}_{w_1}$$

- for each $f \in \Sigma_{w,A}$ a function $f^{\mathcal{M}_{w,A}} : \mathcal{M}_w \to \mathcal{M}_A$, such that

$$f \in \Sigma_{w_1,A_1} \cap \Sigma_{w_2,A_2} \quad \text{and} \quad w_1 \leq w_2 \quad \text{imply} \quad f^{\mathcal{M}_{w_1,A_1}} \text{ is } f^{\mathcal{M}_{w_2,A_2}} \text{ restricted to } \mathcal{M}_{w_1}$$

If $\mathcal{M}$ is a model for $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$ and $\mathcal{L}'$ is an expansion of $\mathcal{L}$ then an *expansion* of $\mathcal{M}$ to $\mathcal{L}'$ is a model of $\mathcal{L}'$ with the same universe as $\mathcal{M}$ which agrees with $\mathcal{M}$ on the symbols in Σ.

A *homomorphism* $h : \mathcal{M} \to \mathcal{N}$ between models $\mathcal{M}$ and $\mathcal{N}$ is an $\mathcal{S}$-sorted family of functions

$\{h_A : \mathcal{M}_A \to \mathcal{N}_A \mid A \in \mathcal{S}\}$ satisfying the following conditions (suppressing sort information for readability).

$$A \leq A' \text{ implies } h_A = (h_{A'}) \restriction_{\mathcal{M}_A}$$
$$h(f^{\mathcal{M}}(a_1, \ldots, a_n) = f^{\mathcal{N}}(h(a_1), \ldots, h(a_n)), \text{ and}$$
$$R^{\mathcal{M}}(h(a_1, \ldots, a_n)) \text{ implies } R^{\mathcal{N}}(h(a_1), \ldots, h(a_n))$$

4.2.0.1 Motivating (local) filtering

Two cautionary examples from [GM92].

Example 2. *$A \leq B$, $A < C$, $f \in \Sigma_{B,B} \cap \Sigma_{C,C}$. Let $\mathcal{M}$ have $\mathcal{M}_A = \{a\}$, $\mathcal{M}_B = \{a, b\}$, $\mathcal{M}_C = \{a, c\}$, let $f^{\mathcal{M}_{B,B}}$ be the constant function b; let $f^{\mathcal{M}_{C,C}}$ be the constant function c. Thus these two functions, each interpreting f, do not agree on a.*

Filtering and the monotonicity condition on models will preclude this.

Example 3. *[GM92]] $B \leq A, B \leq C$; $a : A, b : B, c : C$; postulate $a = c$.*

The term algebra does not satisfy $a = c$. But the algebra that interprets A as $\{d, e\}$, C as $\{d, e\}$, B as $\{d\}$ with constant b interpreted as d and constants a, c both interpreted as e, does satisfy it — and this alg is isomorphic to the term algebra! So equations are not preserved under isomorphism.

Assumption of local filtering precludes this phenomenon.

4.2.0.2 The Term Model

When the set of relation symbols in $\mathcal{L}$ is empty then the set of closed terms forms the universe of a model for $\mathcal{L}$, the *term algebra* [GM92]. We may view this as a model for an arbitrary order-sorted signature, as follows.

Definition 1. *Fix $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$. The family $\mathcal{T}^{\mathcal{L}} = \{\mathcal{T}^{\mathcal{L},A} \mid A \in \mathcal{S}\}$ of* closed terms *over $\mathcal{L}$ is the $\subseteq$- least family satisfying*

- $\Sigma_{\langle\rangle,A} \subseteq \mathcal{T}^{\mathcal{L},A}$;
- *if $f \in \Sigma_{w,s}$ with $w = A_1 \ldots A_n$ and for each i, $t_i \in \mathcal{T}^{\mathcal{L},A_i}$ then $f(t_1, \ldots, t_n) \in \mathcal{T}^{\mathcal{L},A}$;*
- *is $A \leq A'$ then $\mathcal{T}^{\mathcal{L},A} \subseteq \mathcal{T}^{\mathcal{L},A'}$.*

The family $\{\mathcal{T}^{\mathcal{L},A} \mid A \in \mathcal{S}\}$ determines a model $\mathcal{T}^{\mathcal{L}}$ of $\mathcal{L}$, the term model*, by taking the interpretation $f^{\mathcal{T}^{\mathcal{L}}}$ of each $f \in \Sigma_{\langle A_1 \ldots A_n\rangle,A}$ to be the function taking each tuple $(t_1, \ldots, t_n) \in (\mathcal{T}^{\mathcal{L},A_1} \times \cdots \times \mathcal{T}^{\mathcal{L},A_n})$ to the term $f(t_1, \ldots, t_n)$, and taking the interpretation of each relation symbol to be the empty relation.*

The following result is an easy consequence of the initiality of term models in order-sorted algebra, but it is crucial to our development.

Theorem 1. *Suppose $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$ is a regular signature such that Σ has no relation symbols. Then the term model $\mathcal{T}^{\mathcal{L}}$ is initial. That is, for any model $\mathcal{M}$ of $\mathcal{L}$ there is a unique homomorphism from $\mathcal{T}^{\mathcal{L}}$ to $\mathcal{M}$.*

Proof. Initiality of $\mathcal{T}^{\mathcal{L}}$ in the category of *algebras* was shown by Goguen and Meseguer [GM92]. Now, given an $\mathcal{L}$-model $\mathcal{M}$, we let $\mathcal{M}'$ be the reduct of $\mathcal{M}$ to $\mathcal{L}'$, is the reduct of $\mathcal{L}$ obtained by removing the relation symbols: $\mathcal{M}'$ is a $\mathcal{L}$'-algebra. The unique algebra homomorphism from $\mathcal{T}^{\mathcal{L}}$ to $\mathcal{M}'$ is itself a $\mathcal{L}$-homomorphism from $\mathcal{T}^{\mathcal{L}}$ to $\mathcal{M}$, simply because each $\mathcal{T}^{\mathcal{L}}$-relation is empty. □

Here's why we want a cross-domain equality predicate (rather than each sort having its own equality predicate).

Example 4 ([GM92])**.** *$A \leq B, C \leq B, C \leq D$, with $a : A$ etc. Equations $a = b$, $b = c$, $c = d$ want to imply $a = d$ by transitivity but A and D are incomparable. Leads to allowing equations between terms in same connected component.*

Summary: we allow equations between terms in the same connected component (in order to support transitivity). But we may assume terms in an equation have the same sort (Definition 3) by local filtering.

4.2.1 Formulas and Truth

Henceforth we assume that our signatures are coherent: regularity ensures that the term model is initial, and local filtering allows us to assume that terms in an equation have the same type common type). (even though we have "cross-sort" equality, terms in the same connected component of the sort poset do have a common type).

Definition 2 (Open terms)**.** *Fix a signature $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$. Let X be an $\mathcal{S}$-sorted set $\{X_A \mid A \in \mathcal{S}\}$ of variables, with the X_A mutually disjoint and disjoint from*

Σ. *The set* $\mathcal{T}^{\mathcal{L}}(X)$ *of* (open) terms over X *is, intuitively, obtained by adjoining the variables in* X_A *to the term model at type* A. *Formally we proceed as follows [GM92]. Define* Σ^X *to be the family of symbols with* ${\Sigma^X}_{\langle\rangle,A} = \Sigma_{\langle\rangle,A} \cup X_A$ *and* ${\Sigma^X}_{w,A} = \Sigma_{w,A}$ *for* $w \neq \langle\rangle$. *Then* $\mathcal{L}^X$ *is the signature* $(\mathcal{S}, \leq, \Sigma^X)$. *Then the family* $\mathcal{T}^{\mathcal{L}}(X)$ *of open terms over* X *is the family* $\mathcal{T}^{\mathcal{L}^X}$ *as defined in Definition 1, that is the terms of* $\mathcal{T}^{\mathcal{L}}(X)$ *are the closed terms over* $\mathcal{L}^X$.

As noted by Goguen and Meseguer [GM92] the fact that the open term algebra is initial in the category of $\mathcal{L}^X$-algebras entails the fact that this algebra is *free* over the generators X in the category of $\mathcal{L}$-algebras. In particular, if $\eta : X \to \mathcal{M}$ is an $\mathcal{S}$-sorted assignment of values in $\mathcal{M}$ to variables from X then there is a canonical way to extend η to map $\mathcal{T}^{\mathcal{L}}(X)$ to $\mathcal{M}$.

Definition 3 (Formulas)**.** *An* atomic formula *is one of*

- $\perp$ *(to be interpreted as falsehood)*
- $P(t_1, ..., t_n)$ *where* $P \in \Sigma_{\langle A_1, ... A_n\rangle}$ *and* $t_i \in \mathcal{T}^{\Sigma, A_i}(X)$ *for all* $1 \leq i \leq n$.
- $t_1 = t_2$ *where for some* A, *for all* i, $t_i \in \mathcal{T}^{\Sigma, A}(X)$.

The formula $\perp$ *will be convenient in Section 4.3. The set of* formulas *is defined inductively by closing the set of atomic formulas under the propositional operators* $\wedge$, $\vee$, *and* $\neg$ *and the quantifiers* $\exists$ *and* $\forall$. *We will indicate quantification over a sorted variable* $x \in X_A$ *by* $\exists x^A$ *or* $\forall x^A$.

The notions of free and bound variable are standard; let $FV(\phi)$ *denote the set of free variables of formula* ϕ. *A* sentence *is a formula with no free variable occurrences.*

We don't need to introduce a constant for "true", as the negation of falsehood, since any sentence of the form $\forall x^A \,.\, x = x$ will serve (as will be seen, existential-free

sentences will play a key role in the following, and $\bot$ provides an existential-free falsehood).

Definition 4. *An environment η over a model $\mathcal{M}$ is an $\mathcal{S}$-indexed family of partial functions $\{\eta_s : X_s \to \mathcal{M}_A \mid A \in \mathcal{S}\}$ such that $\eta_A = (\eta_{A'})|_{X_A}$ whenever $A \leq A'$. As usual the notation $\eta[x^A \mapsto e]$ refers to the environment agreeing with η except that it maps variable $x \in X_A$ to e. An environment η can be extended to terms in $\mathcal{T}^{\Sigma}(X)$ in the usual way.*

Definition 5 (Truth in a model)**.** *Let $\mathcal{M}$ be a model, ϕ a formula, and η an environment such that $FV(\phi) \subseteq dom(\eta)$. The relation $\mathcal{M} \models_\eta \phi$ is defined by induction over ϕ as follows.*

- *If ϕ is $\bot$ then $\mathcal{M} \models_\eta \phi$ fails.*
- *If $\phi \equiv P(t_1, ..., t_n)$ then $\mathcal{M} \models_\eta \phi$ if and only if $P^{\mathcal{M}}(\eta(t_1), ..., \eta(t_n))$ holds.*
- *If $\phi \equiv t_1 = t_2$ then $\mathcal{M} \models_\eta \phi$ if and only if $\eta(t_1) = \eta(t_2)$ holds.*
- *If $\phi \equiv \neg\alpha$ then $\mathcal{M} \models_\eta \phi$ if and only if $\mathcal{M} \not\models_\eta \alpha$.*
- *If $\phi \equiv \alpha \wedge \beta$ then $\mathcal{M} \models_\eta \phi$ iff $\mathcal{M} \models_\eta \alpha$ and $\mathcal{M} \models_\eta \beta$.*
- *If $\phi \equiv \alpha \vee \beta$ then $\mathcal{M} \models_\eta \phi$ iff $\mathcal{M} \models_\eta \alpha$ or $\mathcal{M} \models_\eta \beta$.*
- *If $\phi \equiv \exists x^A \alpha$ then $\mathcal{M} \models_\eta \phi$ iff there is some element e in $\mathcal{M}_A$ such that $\mathcal{M} \models_{\eta[x \mapsto e]} \alpha$.*
- *If $\phi \equiv \forall x^A \alpha$ then $\mathcal{M} \models_\eta \phi$ iff for each element e in $\mathcal{M}_A$ it holds that $\mathcal{M} \models_{\eta[x \mapsto e]} \alpha$.*

We allow empty sorts in our models, indeed we even allow all sorts to be empty, and it is well-known that the possibility of empty sorts introduces subtleties into

formal *deduction* [SNGM89]. But we are not interested in formal deduction *per se* and, given the fact that we require that the domain of an environment include all the free variables of a formula under consideration, there are no semantic difficulties arising from empty sorts. For example, if sort A is empty in a model then any environment η must have η_A be the empty function; as a consequence any formula $\exists x^A.\phi$ will be false under η, and any formula $\forall x^A.\phi$ will be true under η.

4.2.1.1 On reduction to unsorted logic

It is not unusual for treatments of many-sorted logic to "encode" sorts as unary predicates and to view many-sorted logic as a particular syntactic discipline over standard one-sorted logic. For example one can reasonably view the sorted quantification $\exists x^A\ .\ \phi$ as shorthand for $\exists x\ .\ (A(x) \wedge \phi)$. This is the traditional approach taken in mathematical logic [End72]. To handle subsorting is is natural to introduce *coercion functions:* to capture $A \leq B$ in the order-sorted setting we can add to the signature a function symbol $c_{AB} : A \to B$ in the flat setting. There are some natural axioms imposed on the coercion functions: that they are injective, that they compose properly to reflect transitivity, and so forth.

Definition 6. *Let* $(\mathcal{S}, \leq, \Sigma)$ *be a signature. The* flat many-sorted encoding *of* $\mathcal{L}$ *is the signature* $\mathcal{L}^+ = (\mathcal{S}, \emptyset, \Sigma^+)$ *where*

$$\Sigma^+ = \Sigma \cup \{c_{A,A'} \in \Sigma^+_{A,A'} \mid A \leq A'\}.$$

There are some natural axioms on the coercion functions.

1. $c_{A,A'}(x) = x$
2. $c_{A,A'}$ is injective

3. if $A \leq A' \leq A''$ then $c_{A',A''} \circ c_{A,A'} = c_{A,A''}$

4. each c commutes with functions in Σ.

The Reduction Theorem of [GM92]:

Theorem 2 ([GM92])**.** *When* $(\mathcal{S}, \leq, \Sigma)$ *is regular and locally filtered then order-sorted algebra can be reduced to ordinary many-sorted algebras satisfying the axioms above via an equivalence of categories.*

This theorem lifts, of course, to our setting of order-sorted *logic.* But, for a variety of reasons, we resist such an encoding into unsorted logic. First, it would make counting terms, our central concern, more difficult. This is because the closed terms would involve the coercion functions and we would have to count *modulo* the axioms governing them, otherwise we would overestimate the size of the true set of closed terms in the original signature. Second, introducing coercion functions would clutter the treatment of results such as Herbrand's Theorem. Finally, we want our results to provide clear information to users of tools—like Alloy—that are explicitly order-sorted, and an encoding into another formalism would be an obstacle to these users. So we prefer to work with order-sorted logic directly.

4.3 Skolemization

4.3.1 Negation-normal form

A formula is in *negation-normal* form if the negation sign is applied only to atomic formulas.

Lemma 1. *Every formula is logically equivalent to a formula in negation normal form.*

Proof. As for standard one-sorted logic. DeMorgan's laws for pushing negations below $\wedge$ and $\vee$, and the equivalences between $\neg\exists x^A\alpha$ and $\forall x^A\neg\alpha$ all hold, even in the presence of empty sorts. □ □

Failure of prenex-normal form The fact that models can have empty sorts changes the rules for how quantifiers may be moved within a formula. In particular the classical equivalence

$$((\exists x^A\alpha) \vee \beta) \text{ is equivalent to } \exists x^A(\alpha \vee \beta) \quad [x \text{ not free in } \beta]$$

does not hold if A can be empty (and of course the dual equivalence involving $\forall$ fails as well) and so we cannot in general percolate quantifiers to the front of a formula. So we cannot restrict our attention to formulas in prenex normal form, but we will always pass to negation-normal form.

Definition 7 (Skolemization)**.** *Let ϕ be a negation-normal form formula over signature $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$; the result of a* Skolemization-step *of ϕ is any formula ϕ' that can be obtained as follows. If $\exists x^A.\phi(x^A, x_1^{A_1}, \ldots, x_n^{A_n})$ is a subformula occurrence of ϕ that is not in the scope of an existential quantifier, let f be a function symbol not in Σ, and let ϕ' be the result of replacing the occurrence of $\exists x^A.\phi(x, x_1, \ldots, x_n)$ by $\phi(f(x_1, \ldots, x_n), x_1, \ldots, x_n)$. Note that ϕ' is a formula in an expanded signature obtained by adding f to $\Sigma_{\langle A_1,\ldots,A_n\rangle,A}$.*

A Skolemization *of a formula ϕ is a sentence with no existential quantifiers, obtained from ϕ by a sequence of such steps.*

It is not hard to see that any two Skolemizations of a sentence will differ only in the names of the new function symbols used. We do not need this result here and so will not prove it. But in order to unambiguously speak of *the* Skolemization

of a sentence σ let us agree that we will eliminate existential formulas from left-to-right and use a canonical well-ordering of the universe of potential vocabulary symbols. With this understanding, if σ is a sentence over $\mathcal{L}$ we we will speak of "the Skolemization" of σ, and denote it σ_{sk}.

Lemma 2. *For any σ we have $\sigma_{sk} \models \sigma$.*

Proof. Note that the signature of σ_{sk} is an expansion of the signature of σ so the entailment claim makes sense. It suffices to show that the result of a single Skolem-step on σ entails σ; this is very easy to see from the definition. □ □

In contrast to the classical case we do not have the fact that " σ satisfiable implies σ_{sk} satisfiable." That holds in one-sorted logic because we can always expand a model of σ to properly interpret the Skolem functions and make σ_{sk} true, but this expansion is not always possible in the presence of empty sorts.

Example 5. *Let σ be $(\exists x^A \,.\, (x = x) \vee \exists y^B \,.\, (y = y)\,) \wedge (\forall z^A \,.\, (z \neq z))$. Then σ is satisfiable but its Skolemization $((a = a) \vee (b = b)) \wedge (\forall z^A \,.\, (z \neq z))$ is not.*

We first note that the phenomenon in Example 5 is essentially the only thing that can go wrong: models can be expanded to interpret Skolem functions if we do not existentially quantify over empty sorts.

Definition 8. *A model $\mathcal{M}$ is* safe for *formula ϕ if for every occurrence of a subformula $\exists x^A \,.\, \alpha$ in ϕ we have $\mathcal{M}_A \neq \emptyset$.*

Lemma 3. *If $\mathcal{M} \models \sigma$ and $\mathcal{M}$ is safe for σ then there is an expansion $\mathcal{M}^*$ of $\mathcal{M}$ to the signature of σ_{sk} such that $\mathcal{M}^* \models \sigma_{sk}$.*

Proof. It suffices to show that the corresponding result holds for a single Skolem-step on σ entails σ, so suppose σ and σ' are as in Definition 7. The argument is just

as for classical one-sorted logic: we can expand the model $\mathcal{M}$ to interpret the new function symbol f precisely because $\mathcal{M}$ satisfies the the original σ, but we must know that A is non-empty in case the truth of $\exists x^A.\sigma(x, x_1, \ldots, x_n)$ is needed for this. □ □

This points the way to recovering a weak version of the classical equi-satisfiability result which will be good enough for our present purposes.

Definition 9. *Let ϕ be a formula. An* approximation *of ϕ is a formula obtained by replacing some (zero or more) subformulas $\exists x^A \,.\, \alpha$ of ϕ by $\bot$.*

Lemma 4. *If $\sigma_\bot$ is an approximation of σ then $\sigma_\bot \models \sigma$.*

Proof. We prove by induction over arbitrary formulas ϕ and approximations $\phi_\bot$, and for arbitrary models $\mathcal{M}$ and environments η, if $\mathcal{M} \models_\eta \phi_\bot$ then $\mathcal{M} \models_\eta \phi$. Suppose $\mathcal{M} \models_\eta \phi_\bot$.

- ϕ is a literal: then $\phi_\bot = \phi$. So certainly $\mathcal{M} \models_\eta \phi$.
- $\phi \equiv \alpha \vee \beta$: then $\phi_\bot = \alpha_\bot \vee \beta_\bot$, where $\alpha_\bot$ and $\beta_\bot$, are approximations of α and β. Then $\mathcal{M} \models_\eta \alpha_\bot$ or $\mathcal{M} \models_\eta \beta_\bot$ so by induction $\mathcal{M} \models_\eta \alpha$ or $\mathcal{M} \models_\eta \beta$, as desired.
- $\phi \equiv \alpha \wedge \beta$: then $\phi_\bot = \alpha_\bot \wedge \beta_\bot$, where $\alpha_\bot$ and $\beta_\bot$, are approximations of α and β. Then $\mathcal{M} \models_\eta \alpha_\bot$ and $\mathcal{M} \models_\eta \beta_\bot$ so by induction $\mathcal{M} \models_\eta \alpha$ and $\mathcal{M} \models_\eta \beta$, as desired.
- $\phi \equiv \forall\, x^A \alpha$: then $\phi_\bot = \forall x^A \,.\, \alpha_\bot$, where $\alpha_\bot$ is an approximation of α. For every $e \in \mathcal{M}_A$ we have $\mathcal{M} \models_{\eta[x^A \mapsto e]} \alpha_\bot$, so by induction at each such e we have $\mathcal{M} \models_{\eta[x^A \mapsto e]} \alpha$, so $\mathcal{M} \models_\eta \forall x^A \,.\, \alpha$.

- $\phi \equiv \exists\, x^A \alpha$: then either $\phi_\perp = \exists x^A.\alpha_\perp$ or $\phi_\perp = \perp$. In the former case we have, for some $e \in \mathcal{M}_A$, $\mathcal{M} \models_{\eta[x^A \mapsto e]} \alpha_\perp$, so by induction $\mathcal{M} \models_{\eta[x^A \mapsto e]} \alpha$, so $\mathcal{M} \models_\eta \exists x^A \,.\, \alpha$. The latter case cannot arise under the hypothesis that $\mathcal{M} \models_\eta \phi_\perp$.

□ □

We can now prove a slightly weaker version of the traditional result on preservation of satisfiability.

Lemma 5. *If $\mathcal{M} \models \sigma$ then there is an approximation $(\sigma_\perp^{\mathcal{M}})$ of σ such that $\mathcal{M} \models \sigma_\perp^{\mathcal{M}}$ and $\mathcal{M}$ is safe for $\sigma_\perp^{\mathcal{M}}$.*

Proof. The sentence $\sigma_\perp^{\mathcal{M}}$ is obtained by replacing $\exists x^A \,.\, \alpha$ by $\perp$ precisely when $\mathcal{M}_A = \emptyset$.

Formally we inductively define approximations $\phi_\perp^{\mathcal{M}}$ for arbitrary formulas ϕ as follows.

- ϕ is a literal: then $\phi_\perp^{\mathcal{M}} = \phi$.
- $\phi \equiv \exists\, x^A \alpha$ and $\mathcal{M}_A = \emptyset$: then $\phi_\perp^{\mathcal{M}} = \perp$
- $\phi \equiv \exists\, x^A \alpha$ and $\mathcal{M}_A \neq \emptyset$: then $\phi_\perp^{\mathcal{M}} = \exists x^A.\alpha_\perp^{\mathcal{M}}$.
- $\phi \equiv \forall\, x^A \alpha$: then $\phi_\perp^{\mathcal{M}} = \forall x^A \,.\, \alpha_\perp^{\mathcal{M}}$.
- $\phi \equiv \alpha \vee \beta$: then $\phi_\perp^{\mathcal{M}} = \alpha_\perp^{\mathcal{M}} \vee \beta_\perp^{\mathcal{M}}$.
- $\phi \equiv \alpha \wedge \beta$: then $\phi_\perp^{\mathcal{M}} = \alpha_\perp^{\mathcal{M}} \wedge \beta_\perp^{\mathcal{M}}$.

It is clear from the construction that $\mathcal{M}$ is safe for $\phi_\perp^{\mathcal{M}}$. We now claim that for an arbitrary environment η, if $\mathcal{M} \models_\eta \phi$ then $\mathcal{M} \models_\eta \phi_\perp^{\mathcal{M}}$. But this is a straightforward induction over formulas ϕ. The lemma follows by taking ϕ to be σ. □ □

Lemma 6. *If σ is satisfiable then there exists an approximation $\sigma_\perp$ of σ such that $\sigma_{\perp sk}$ is satisfiable.*

Proof. By Lemma 5 and Lemma 3. □ □

4.4 A Finite Model Theorem for Order-Sorted Logic

Model $\mathcal{M}$ is a *submodel* of model $\mathcal{N}$ if for each A, $\mathcal{M}_A \subseteq \mathcal{N}_A$ and each $f^{\mathcal{M}}$ and $R^{\mathcal{M}}$ are the restrictions of $f^{\mathcal{N}}$ and $R^{\mathcal{N}}$ to $\mathcal{M}$ and $\mathcal{M}$, respectively. Note that we use "submodel" in this strong sense rather than just requiring each $R^{\mathcal{M}}$ to be a subset of $R^{\mathcal{N}}$ (as is done by some authors).

4.4.1 Homomorphisms and Submodels

If $X = \{X_A) \mid A \in \mathcal{S}\}$ is a family of sets with $X_A \subseteq \mathcal{M}_A$ for each $A \in \mathcal{S}$ then we say that X is closed under a function $g : \mathcal{M}_{A_1} \times \cdots \times \mathcal{M}_{A_n} \to \mathcal{M}_A$ if whenever $(a_1, \ldots, a_n) \in X_1 \times \cdots \times X_n$ we have $g(a_1, \ldots, a_n) \in X_A$. Note that this is a stronger claim than saying that the single set $\bigcup X$ is closed under g.

Lemma 7. *Let $h : \mathcal{P} \to \mathcal{M}$ be a homomorphism between models of $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$. There is a unique submodel of $\mathcal{M}$ with universe $\{h_A(\mathcal{P}_A) \mid A \in \mathcal{S}\}$.*

Proof. It is easy to check that the family $\{h_A(\mathcal{P}_A) \mid A \in \mathcal{S}\}$ is closed under the interpretations in $\mathcal{M}$ of the function symbols in Σ. So if we define the interpretations of the relation symbols in Σ to be the restriction of the interpretations in $\mathcal{M}$ the result is a submodel. Since there is no choice in the interpretations of the symbols in Σ once the universe $\{h_A(\mathcal{P}_A) \mid A \in \mathcal{S}\}$ is determined, uniqueness follows. □ □

We will denote the submodel identified in Lemma 7 as $h(\mathcal{M})$.

Remark 1. *For future reference we observe that if $\mathcal{P}$ is a submodel of $\mathcal{M}$ and $e \in \mathcal{M}_B$ then it need not be the case that $e \in \mathcal{P}_B$ even if $e \in \bigcup\{\mathcal{P}_A \mid A \in \mathcal{S}\}$. Indeed this can happen even when $\mathcal{P}$ is obtained as the image of a homomorphism into $\mathcal{M}$. This has important consequences for the use of sorts as predicates, as we will discuss in Section 4.6.*

Next we establish the fundamental fact about preservation of universal sentences under submodel.

Theorem 3. *Let σ be a sentence that is existential-free and in negation-normal form and let $\mathcal{M}'$ be a submodel of $\mathcal{M}$. If $\mathcal{M} \models \sigma$ then $\mathcal{M}' \models \sigma$.*

Proof. We prove the following for arbitrary formulas ϕ: *for any environment η over $\mathcal{M}'$, if $\mathcal{M} \models_\eta \phi$ then $\mathcal{M}' \models_\eta \phi$.* Suppose ϕ is $P(t_1, ..., t_n)$ or $\neg P(t_1, ..., t_n)$, where P is a relation symbol from Σ or the equality symbol. Each $\eta(t_i)$ is in the domain of $\mathcal{M}'$, so we have that $\mathcal{M} \models_\eta P(t_1, ..., t_n)$ iff $\mathcal{M}' \models_\eta P(t_1, ..., t_n)$ by definition of submodel.

Proceeding inductively: when ϕ is $\alpha \wedge \beta$ or ϕ is $\alpha \vee \beta$ the result is an immediate application of the induction hypothesis. Finally suppose that ϕ is $\forall x^A \alpha$. Then $\mathcal{M} \models \phi$ implies that $\mathcal{M} \models_{\eta[x^A \mapsto e]} \alpha$ for all $e \in \mathcal{M}_A$. Since $\mathcal{M}_A \supseteq \mathcal{M}'_A$ we have $\mathcal{M} \models_{\eta[x^A \mapsto e]} \alpha$ for all $e \in \mathcal{M}'_A$, that is, $\mathcal{M}' \models_\eta \forall x^A \alpha$. □ □

We pause here to point out that Theorem 3 *fails* if sort names are permitted to be used as unary predicates. This is simply because in the definition of submodel there is no requirement that, for example, if element e lies in sort A in a model then e is in sort A in the submodel. This issue is discussed in detail in Section 4.6.

4.4.2 The Kernel of a Model

Definition 10 (The kernel of a model)**.** *Let $\mathcal{M}$ be a model for the regular signature $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$. Let h be the unique homomorphism from $\mathcal{T}^{\mathcal{L}}$ to $\mathcal{M}$ (c.f. Theorem 1). The image of h is a submodel of $\mathcal{M}$ by Lemma 7; this is the kernel of $\mathcal{M}$.*

The crucially important fact for us is that for the kernel $\mathcal{K}$ of $\mathcal{M}$ we have, for each sort A, the cardinality of $\mathcal{K}_A$ is bounded by the the cardinality of ${\mathcal{T}^{\mathcal{L}}}_A$, simply because $\mathcal{K}_A$ is the image of ${\mathcal{T}^{\mathcal{L}}}_A$ under h.

4.4.2.1 The kernel and the Skolem hull

Recall the classical treatment of Skolemization (see *e.g.*, [CK73]): given a model $\mathcal{M}$, let $\mathcal{M}^*$ be a Skolem expansion, *i.e.* a model interpreting the Skolem functions, that satisfies the Skolem theory (the sentences saying that the Skolem functions witness the truth of the associated existential formula). Then given a subset X of the universe of $\mathcal{M}$, the Skolem hull $H_{\mathcal{M}}(X)$ is the smallest subset of the universe containing X and closed under the functions and constants of the enriched language; this determines an elementary submodel $\mathcal{H}_{\mathcal{M}}(X)$ of $\mathcal{M}$. In particular $\mathcal{H}_{\mathcal{M}}(\emptyset)$ can be viewed as a "minimal" submodel of $\mathcal{M}$.

But in the order-sorted setting, *the kernel of a model is not in general the same as the Skolem hull.* The latter notion, although perfectly sensible in order-sorted logic, does not play the same role of "minimal" submodel as it does in the one-sorted setting. Indeed it is possible for the kernel of a model to be finite while the Skolem hull is infinite.

Example 6. *Consider $\mathcal{L} = (\{A, B\}, \emptyset, \Sigma)$ with $a \in \Sigma_{\langle\rangle, A}$ and $f \in \Sigma_{B,B}$ the only vocabulary symbols. Let $\mathcal{M}$ have $\mathcal{M}_A = \{b_0 = a^{\mathcal{M}}\}$, $\mathcal{M}_B = \{b_0, b_1, b_2, \ldots\}$, and $f^{\mathcal{M}}$ map b_i to b_{i+1}. Then the Skolem hull $\mathcal{H}(\emptyset)$ of $\mathcal{M}$ is $\mathcal{M}$ itself. Yet the kernel $\mathcal{K}$ of*

$\mathcal{M}$ is the model of size 1 with $\mathcal{K}_A = \{b_0\}$, $\mathcal{K}_B = \emptyset$, $f^{\mathcal{K}} = \emptyset$.

4.4.3 A Finite Model Theorem

Here we present our main theorem.

Theorem 4. *Let σ be an $\mathcal{L}$-sentence whose Skolemization σ_{sk} has signature $\mathcal{L}^*$. Then σ is satisfiable if and only if σ has a model $\mathcal{H}$ such that for each sort A, the cardinality of $\mathcal{H}_A$ is no greater than the cardinality of ${\mathcal{T}^{\mathcal{L}^*}}_A$.*

Proof. For the non-trivial direction, suppose σ is satisfiable. By Lemma 6 there is an approximation $\sigma_\perp$ of σ such that $(\sigma_\perp)_{sk}$ is satisfiable. Let $\mathcal{L}^{**}$ be the signature for $\sigma_{\perp sk}$; note that $\mathcal{L}^{**}$ is a reduct of $\mathcal{L}^*$ and the sentence $(\sigma_\perp)_{sk}$ is existential-free.

Let $\mathcal{M}$ be a model of $(\sigma_\perp)_{sk}$, and let $\mathcal{H}$ be the kernel of $\mathcal{M}$. Since $(\sigma_\perp)_{sk}$ is existential-free, $\mathcal{H} \models (\sigma_\perp)_{sk}$. Since $\mathcal{H}$ is a kernel we have that for each sort A, the cardinality of $\mathcal{H}_A$ is no greater than the cardinality of ${\mathcal{T}^{\mathcal{L}^{**}}}_A$, and thus no greater than the cardinality of ${\mathcal{T}^{\mathcal{L}^*}}_A$. Since $(\sigma_\perp)_{sk} \models \sigma_\perp$ and $\sigma_\perp \models \sigma$, the model $\mathcal{H}$ is the desired model of σ. □ □

Finally we can define precisely the key notion of the paper.

Definition 11. Order-Sorted Effectively Propositional Logic (OS-EPL) *is the class of sentences σ such that the signature of the Skolemization of σ has a finite term model.*

In Section 4.5 we will show how to decide whether a sentence is in OS-EPL and if so, to compute the sizes of the sorts in the term model. Taken together with Theorem 4, this establishes a decision procedure for satisfiability of OS-EPL sentences.

4.4.4 Herbrand's Theorem

As a brief digression, we address Herbrand's Theorem for order-sorted logic. The standard model-theoretic proof of Herbrand's Theorem in first-order logic uses in an essential way the fact that every element in the Skolem hull of a model is named by a term. As we have noted this fails when sorts may overlap. There are proof-theoretic approaches to Herbrand's Theorem of course but proof theory of order-sorted logic, especially in the presence of empty sorts, is delicate, so a model-theoretic proof would be nice to have. We are in a position to give such a proof here.

Theorem 5 (Herbrand's Theorem for Order-Sorted Logic)**.** *Let $\tau \equiv \forall y_1, \ldots, y_n \,.\, \alpha$ be a sentence with α quantifier-free. Then τ is unsatisfiable iff there is a set $\{\alpha_1, \ldots, \alpha_k\}$ of closed instances of α such that $(\alpha_1 \wedge \cdots \wedge \alpha_k)$ is unsatisfiable.*

Proof. It is convenient to prove the following expanded version of the theorem. The following are equivalent.

1. $\tau = \forall y_1 \ldots y_n \,.\, \alpha$ is satisfiable.
2. The set $T = \{\alpha_* \mid \alpha_* \text{ is a closed instance of } \alpha\}$ is satisfiable.
3. For every finite subset $\{\alpha_1, \ldots, \alpha_n\}$ of closed instances of α, $\quad (\alpha_1 \wedge \cdots \wedge \alpha_n)$ is satisfiable.

The implication 1 to 2 is immediate, since any model of τ will satisfy T. To see that 2 implies 1: let $\mathcal{M} \models T$. Let $\mathcal{M}_0$ be the kernel of $\mathcal{M}$. So $\mathcal{M}_0 \models T$ by Theorem 3. Then $\mathcal{M}_0 \models \tau$ because the universal quantifier just ranges over closed terms when interpreted in $\mathcal{M}_0$.

The equivalence of (2) and (3) is just the Compactness Theorem *for ordinary propositional logic.* □ □

It is worth noting that we do not in the above proof, require the Compactness Theorem for order-sorted logic *per se*.

4.5 Algorithms

In this section we present an algorithm to determine, given a signature $\mathcal{L} = (\mathcal{S}, \leq .\Sigma)$ which sorts of $\mathcal{S}$ are inhabited by only finitely many closed terms, and an algorithm to count the number of closed terms inhabiting a sort.

Notation Fix a signature $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$. We say that sort A is *finitary* in $\mathcal{L}$ if $\mathcal{T}^{\mathcal{L},A}$ is finite.

4.5.1 Testing OS-EPL membership

Definition 12. *Let $\mathcal{L} = (\mathcal{S}, \Sigma)$ be a many-sorted signature. The grammar $G_{\mathcal{L}}$ is defined as follows. The set of nonterminals is $\mathcal{S} \cup \{A_0\}$, where A_0 is a fresh symbol not in $\mathcal{S}$, the set of terminals is $\bigcup\{\Sigma_{w.S} \mid (w, s) \in \mathcal{S}^* \times \mathcal{S}\}$, and the set of productions comprises:*

$$\begin{aligned} A_0 &\to A && \text{for each } A \in \mathcal{S} \\ A &\to a && \text{whenever } a \in \Sigma_{\langle\rangle,A} \\ B &\to f A_1 \dots A_n && \text{whenever } f \in \Sigma_{\langle A_1 \dots A_n\rangle,B} \\ B &\to A && \text{whenever } A \leq B \end{aligned}$$

Recall that a non-terminal X in a CFG G is said to be *useful* if there exists a derivation $A_0 \Rightarrow^* \alpha X \beta \Rightarrow^* u$ where u is a string of terminals, otherwise X is *useless.* If A is a useful non-terminal and u is a string of terminals we say that A *generates*

u if there is a derivation $A \Longrightarrow^* u$.

Lemma 8. *Let A be a sort of $\mathcal{L}$ and let u be a string of terminals over $\bigcup\{\Sigma_{w.S} \mid (w, s) \in \mathcal{S}^* \times \mathcal{S}\}$. Then u is a term in $\mathcal{T}^{\mathcal{L},A}$ if and only if there is a derivation $A \Rightarrow^* u$ in $G_{\mathcal{L}}$. A sort A is inhabited by closed term if and only if A is useful in the grammar $G_{\mathcal{L}}$. When A is useful as a sort in $L(G_{\mathcal{L}})$, the set $(\mathcal{T}^{\mathcal{L}})_A$ is finite if and only if A generates only finitely many terms in in $L(G_{\mathcal{L}})$. In particular the set $\mathcal{T}^{\mathcal{L}}$ is finite if and only if $L(G_{\mathcal{L}})$ is finite.*

Proof. The first claim is easy to check: it holds essentially by the construction of $G_{\mathcal{L}}$. The second claim follows from the first and the facts that the u in question are strings of terminals of $G_{\mathcal{L}}$ and we have $A_0 \Rightarrow A$ for each $A \in \mathcal{S}$. □ □

Theorem 6. *There is an algorithm that, given an order-sorted signature $\mathcal{L}$, determines (uniformly) for each sort A, whether ${\mathcal{T}^{\mathcal{L}}}_A$ is finite. The algorithm runs in time linear in the total size of $\mathcal{L}$.*

Proof. By Lemma 8, ${\mathcal{T}^{\mathcal{L}}}_A$ is finite if and only if A generates only finitely many terms in in $L(G_{\mathcal{L}})$. There is a well-known algorithm for testing whether a non-terminal in a context-free grammar generates infinitely many terminal strings [HMU06]. Transliterated into our setting the algorithm is as follows. First restrict attention to those sorts A that are inhabited by closed strictly-typed terms (*i.e.* eliminate "useless symbols" from the grammar $G_{\mathcal{L}}$): this can be done in linear time with a judicious choice of data structures (see for example [HMU06]). Next, form the graph whose nodes are the inhabited sorts, with an edge from B to A if and only if there is a production in $G_{\mathcal{L}}$ of the form $B \to \alpha\ A\ \beta$, that is, if and only if the set $\Sigma_{\langle A_1 \ldots A \ldots A_n \rangle, B}$ is non-empty or if $A \leq B$. Having ensured in the previous step that each sort named by a non-terminal in $G_{\mathcal{L}}$ is inhabited, it is the case that A generates infinitely many terminal strings if and only if there is a path from A to a cycle. The set of such

sorts can be checked in linear time by a depth-first search. Since the size of $G_{\mathcal{L}}$ is linear in the size of $\mathcal{L}$, the overall complexity of our algorithm is linear in $\mathcal{L}$. □ □

Example 7. *Return to Example 4.1 from the introduction. Over the signature $\mathcal{L}$ with two sorts A and B, with $A \leq B$, consider the sentence*

$$\forall y_1^A \exists x^B \forall y_2^A \,.\, \phi \tag{4.2}$$

where ϕ has no function symbols. After Skolemizing we have the signature with $b \in \Sigma_{\langle\rangle,B}$ and $f \in \Sigma_{A,B}$ in addition to those constants in the original signature. The corresponding grammar has productions $A_0 \to A$, $A_0 \to B$, $B \to b$, $B \to f\ A$ and $B \to A$, in addition to productions corresponding to the constants appearing in the original ϕ. The resulting graph has edges from the node A_0 to A and to B, and an edge from B to A (the latter for two reasons, due to the grammar production $B \to f\ A$ and due to the production $B \to A$). This graph is acyclic so we conclude that this class of sentences has the finite model property.

On the other hand, if we were to postulate that $B \leq A$ (instead of $A \leq B$) then we cannot deduce the finite model property. Our grammar would have the production $A \to B$ in addition to $B \to A$ and the resulting graph would have a cycle.

Corollary 1. *Membership in OS-EPL is decidable in linear time.*

Proof. Let σ be given, over signature $\mathcal{L}$. We can compute the skolemization σ_{sk} of σ in linear time, and extract the signature $\mathcal{L}^*$ of σ_{sk}. The size of this signature is clearly linear in σ, so by Theorem 6, we can decide whether all sorts of $\mathcal{L}^*$ are finitary in time linear in σ. □ □

4.5.2 Computing the number of terms in a sort

Note that in the worst case, Σ may induce a number of terms exponential in its size. Thus we would like to avoid actually generating the terms, and merely count them if we can do so in polynomial time.

The intuition behind Algorithm 2 is as follows. If a sort is finitary, its terms can be of height no greater than the number of functions in Σ. So we construct a table containing the number of terms of each height of each sort, starting with constants and then applying functions. The only complication is that when counting the ways to create a new term of height h using function f, we need to make certain that each has at least one subterm of height *exactly* $h-1$.

Theorem 7. *There is an algorithm that, given a regular signature $\mathcal{L}$ with no overloading, computes, in time cubic in the size of $\mathcal{L}$, the size of $\mathcal{T}^{\mathcal{L},A}$ for each finitary sort A (returning "∞" for the non-finitary sorts).*

Proof. The algorithm is given as Algorithm 2 below.

Proof of correctness Since the algorithm uses only FOR loops, it is easy to see that it terminates. Furthermore we claim that after termination, the totals of each column $\mathsf{Tbl}[\sum][A]$ contain exactly $|\mathcal{T}^{\mathcal{L},A}|$ for each finitary sort A.

First observe that by the pigeonhole principle, all terms in $\mathcal{T}^{\mathcal{L},A}$ (A finitary) must have *height* $\leq \mathsf{n_f}$. Therefore, when counting terms in finitary sorts it suffices to count only terms of *height* $\leq \mathsf{n_f}$, and thus we need only prove that the algorithm populates the table correctly: that each $\mathsf{Tbl}[h][A]$ contains exactly the number of terms having height h within $\mathcal{T}^{\mathcal{L},A}$.

Proof: After a row is computed by our algorithm, it is never again modified. So we proceed by induction on h.

Algorithm 2: The Counting Algorithm

Input: A signature $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$
Output: For each sort s for which Σ in finitary, the number of terms of sort s.

let $\mathsf{n_s}$ be the number of sorts ;
let $\mathsf{n_f}$ be the number of functions ;
let Tbl be a 2-dimensional vector of size $[\mathsf{n_f} + 1][\mathsf{n_s}]$;
initialize Tbl with 0s ;

```
// Fill the first row with constant term counts
```
for *each constant symbol* $c : S$ **do**
    `// Populate`
    Increment Tbl [0][S] by 1;
    `// Propagate to supersorts`
    for *each sort S' such that* $S < S'$ **do**
        Increment Tbl [0][S'] by 1;

for h = *1 to* $\mathsf{n_f}$ **do**
    foundTermsOfThisHeight := false;
    for *each function* $f : (A_1, ..., A_n) \to B$ **do**
        `// Compute the number of ways to construct a term of height h via f. (To be this height, must use at least one term of height h-1.)`
        ways $= 0$;
        `// Ways to make a term with leftmost h-1 leftmost=`
        for leftmost$=$ *1 to* n **do**
            `// Start with the number of h-1 height terms having this sort`
            $\mathsf{ways}_n = \mathsf{Tbl}[h-1][\mathsf{leftmost}]$;
            `// And multiply by the number of available subterms of the other sorts in f's arity`
            for component$=$ *1 to* $(\mathsf{leftmost} - 1)$ **do**
                $\mathsf{ways}_n * = \sum\{\mathsf{Tbl}[k][\mathsf{component}] \mid 0 \leq k \leq h-2\}$
            for component$=$ $(\mathsf{leftmost} + 1)$ *to* n **do**
                $\mathsf{ways}_n * = \sum\{\mathsf{Tbl}[k][\mathsf{component}] \mid 0 \leq k \leq h-1\}$
            $\mathsf{ways} += \mathsf{ways}_n$
        `// ways contains number of height h terms produced by function f. Add those terms to the table...`
        $\mathsf{Tbl}[h][B] += \mathsf{ways}$;
        if ways > 0 **then**
            foundTermsOfThisHeight := true
        `// ... and propagate to supersorts`
        for *each sort S' such that* $B < S'$ **do**
            $\mathsf{Tbl}[h][S'] += \mathsf{ways}$
    `// No height h terms means no larger terms.`
    if *not foundTermsOfThisHeight* **then**
        break;

return a 1-dimensional vector of size $\mathsf{n_s}$which contains the column sums of Tbl

Base: If $h = 0$ then we are concerned only with constant terms. The first block of our algorithm counts every constant $c : S$ exactly once in each $\mathsf{Tbl}[0][S']$ such that $S \leq S'$. So we can conclude that $\mathsf{Tbl}[0]$ contains a faithful count of height 0 terms for all sorts.

Induction: Suppose $h > 0$ and that each $\mathsf{Tbl}[x]$, $0 \leq x < h$ is correct. A non-constant term t is in $\mathcal{T}^{\mathcal{L},A}$ if (by definition!):

1. t has a function at its head with result sort A
2. t has a function at its head with some result sort B with $B \leq A, B \neq A$.

The algorithm increments a table cell according to case (2) if and only if it has already incremented a cell according to case (1).

Each ground term of height $h > 0$ has one distinct function f at its head, and (with respect to the ordering in f's arity) exactly one left-most subterm of height $h - 1$ at index lm, $1 \leq \mathsf{leftmost} \leq n$. So we only need to show that we correctly calculate the number of terms in $\mathcal{T}^{\mathcal{L},A}$ having height h, head function f with result sort A and left-most $h - 1$ subterm at index $\mathsf{leftmost}$

The number of such terms depends only on the number of subterms available to fill each index of f's arity. The number of usable subterms for A_i is:

- If $i = \mathsf{leftmost}$, terms of sort A_i having height exactly $h - 1$ are admissible.
- If $i < \mathsf{leftmost}$, terms of sort A_i having height up to $h - 2$ are admissible.
- If $i > \mathsf{leftmost}$, terms of sort A_i having height up to $h - 1$ are admissible.

(The usable heights differ by index since index $\mathsf{leftmost}$ is the *leftmost* appearance of a height $h - 1$ subterm.)

But this is exactly the calculation that the algorithm makes, and by our induction hypothesis, these subterm rows have been calculated correctly.

Complexity Note that we could optimize the counting algorithm by memoizing column totals, saving us the trouble of repeatedly summing up $\sum\{\mathsf{Tbl}[k][\mathsf{component}] \mid 0 \leq k \leq h-1\}$ and $\sum\{\mathsf{Tbl}[k][\mathsf{component}] \mid 0 \leq k \leq h-1\}$. The code for this is omitted for clarity, but we assume it when calculating the complexity bounds below.

The initial pass for row 0 takes no more than $\mathsf{n_c n_s}$ steps.

The main block (with memoization) has loop structure as follows:

(1) **for** *each* $h \leq \mathsf{n_f}$ **do**
  (2) **for** *each* f *(*$\leq \mathsf{n_f}$ *iterations)* **do**
    (3) **for** *each lm in f's arity* **do**
      (4) **for** *each component in f's arity not equal to lm* **do**
        (iterations bounded by maximum arity) ...
    (5) **for** *each supersort of f's result sort (*$\leq \mathsf{n_s}$ *iterations)* **do**
      ...

Together (2) and (3) make $|\Sigma|$, the size of the signature. Therefore we have a bound:

$\mathsf{n_c n_s} + \mathsf{n_f}(|\Sigma| * maxarity + \mathsf{n_f n_s})$, which is cubic in the size of the signature.

□ □

We could use this algorithm to test for finitary signatures as well, since if we continue to iterate term height past $|\Sigma_F|$, there will be an increase in $\mathsf{Tbl}[h][S]$ if and only if S is infinitary. However, it benefits us to know *in advance* which sorts are finitary and which functions never produce ground terms, as that may greatly reduce the size of Tbl.

If we want to know the total number of terms across all sorts (without duplication), it is easy enough to add a counter and increment it on population (not propagation) in the algorithm above.

Summarizing, we have the following sound and complete procedure for testing satisfiability of OS-EPL sentences. Given sentence σ, compute its Skolemization σ_{sk}; let $\mathcal{L}^*$ be the signature of σ_{sk}. If the term model $\mathcal{T}^{\mathcal{L}^*}$ is finite then we know that if σ is satisfiable then σ has a model whose universe has cardinalities as given in Theorem 4. Since these bounds are computable we can effectively decide satisfiability for such sentences.

Remark 2. *The results of the algorithm in Theorem 7 can be useful even if not all sorts are finitary. Fontaine and Gribomont [FG03] have implemented an instantiation-based algorithm that takes advantage of the information that certain sorts are guaranteed to have finitely many closed terms. Their algorithm does not do a sophisticated test for this condition, in fact it succeeds only if there are no non-constant terms in the sort in question. Our algorithm here is simple yet will allow their methods to be applicable to a wider class of sentences.*

4.6 About Sorts-as-Predicates

Many formulations of sorted logic, and certain tools, allow sort names to be used as predicate names in formulas. We have not built this into our syntax; in this section we explain why. We start with an example, in which Herbrand's Theorem fails in a dramatic way, with obvious negative consequences for our finite model theorem.

Example 8. *Consider a signature $\mathcal{L} = (\mathcal{S}, \leq, \Sigma)$ with sorts A and B, a constant $b \in \Sigma_{\langle\rangle,B}$ and a function $f \in \Sigma_{A,B}$. Let σ be the following sentence expressing the fact that f is one-to-one but not onto*

$$(\forall x^A \,.\, B(x)) \wedge (\forall y^B \,.\, A(y)) \wedge (\forall x^A \,.\, f(x) \neq b) \wedge (\forall x_1^A x_2^A \,.\, (x_1 \neq x_2) \rightarrow (f(x_1) \neq f(x_2)).$$

Since the first two conjuncts force A and B to be equal, this sentence has only infinite models. But the Herbrand universe for $\mathcal{L}$ is the singleton set $\{b\}$.

What went wrong? The fundamental fact that the truth of existential-free sentences is preserved under submodel, Theorem 3, fails when sorts are allowed as predicates. This is because in the definition of submodel there is no requirement that elements remain in each sort they inhabit in the original model: if $\mathcal{P}$ is a submodel of $\mathcal{M}$ and element e happens to be in $\mathcal{M}_A \cap \mathcal{M}_B$ then it is possible that, say, $\mathcal{P}_A$ but not in $\mathcal{P}_B$. So Theorem 3, crucial to Herbrand's Theorem, as well as to the soundness of our decision procedure, fails at the base case.

A natural response to this might be: if sorts are to used as predicates then the notion of submodel should be refined to reflect this. In particular we might refine the definition of submodel and insist that an element in the universe of a submodel retain all of the "sort-memberships" it had in the original model. Unfortunately, if we do this this something else breaks: the fact that the image of a homomorphism makes a submodel (Lemma 7). Recall that closure under under the functions of a vocabulary is a property of a *family* of sets (*e.g.* the family of images of sorts in the source model) and not the union of this family. When we put a sort-structure on the union, in the target model, of the images of sorts in the source by retaining the sort-memberships in the target the resulting family of sets can fail to be closed under the interpretations of the function symbols.

Example 9. *Refer to Example 6; consider the unique homomorphism $h : \mathcal{K} \to \mathcal{M}$, for which h_A maps b_0 to b_0 and $h_B = \emptyset$. The submodel of $\mathcal{M}$ generated by h interprets sort A as $\{b_0\}$, interprets B as $\emptyset$, and is the universe of a submodel of $\mathcal{M}$ (in which f is interpreted as the empty function). But if we were to insist that element b_0 inhabit sort B in the "submodel" induced by h, then we cannot interpret f: it is supposed to be the restriction of $f^{\mathcal{M}}$ to the universe of our submodel, but*

$f^{\mathcal{M}}(b_0) = b_1$, *and* b_1 *is not in the range of* h.

We stress that these observations are not bound up with our project of trying to build finite models, they are general foundational problems with using sorts as predicates. If we permit sorts to be used as predicates, we must either give up the notion of models being closed under homomorphisms or give up the intuitively compelling model-theoretic result that universal sentences are preserved under submodel.

The solution is to view the use of sorts as predicates as syntactic sugar for formulas in the core language. The construction for de-sugaring is the following. Given σ with subterm $A(t)$ for a sort A, rewrite this to replace $A(t)$ with $(\exists z^A \,.\, z = t)$ (where z is a fresh variable). This is done before passing to negation-normal form, so as a consequence a subformula $\neg A(t)$ will be replaced with $(\forall z^A \,.\, z \neq t)$.

Lemma 9. *If* σ' *is obtained from* σ *by the process described in the previous paragraph then* σ' *and* σ *are logically equivalent.*

Proof. Recall that we define the truth of a formula ϕ in a model $\mathcal{M}$ in the context of environments η under the assumption that the domain of η includes all the free variables of ϕ. Noting that $A(t)$ has the same free variables as $(\exists z^A \,.\, z = t)$ when z is fresh variable, we must show that for any $\mathcal{M}$ and environment η such that the domain of η includes the free variables of t), $\mathcal{M} \models_\eta A(t)$ if and only if $\mathcal{M} \models_\eta (\exists z^A \,.\, z = t)$. The fact that $\mathcal{M}_A$ might be empty does not cause any problems: if $\mathcal{M} \models_\eta A(t)$ this means that $\eta(t) \in \mathcal{M}_A$ and so we can bind z to $\eta(t)$ to witness the truth of $(\exists z^A \,.\, z = t)$; while if $\mathcal{M} \not\models_\eta A(t)$ this means that $\eta(t) \notin \mathcal{M}_A$ and so $\mathcal{M} \models (\forall z^A \,.\, z \neq t)$. □ □

Example 10. *We revisit Example 8. When the sentence there is de-sugared accord-*

ing to the recipe above we arrive at

$$(\forall x^A \exists u^B \,.\, u = x) \wedge (\forall y^B \exists w^A \,.\, w = y) \wedge (\forall x^A \,.\, f(x) \neq b) \wedge (\forall x_1^A x_2^A \,.\, (x_1 \neq x_2) \rightarrow (f(x_1) \neq f(x_2)).$$

When this sentence is Skolemized we get a function from A to B and one from B to A; together with the constant b this obviously generates an infinite set of closed terms.

Summarizing: the use of sorts as predicates is not innocent, but it can be can be accommodated after translation into the core language.

4.6.1 Sorts-as-predicates in Margrave

Of these two equivalent queries: $\exists x^{Subject} Admin(x)$ and $\exists x^{Subject} \exists y^{Admin}(x = y)$, the first is far more succinct and readable. Because of this, Margrave supports a *limited* kind of sort-as-predicate in its query language.

To allow this, we expand the algorithms above (4.5). When Skolemizing a query, Margrave marks whether or not a Skolem function arose due to a sort-as-predicate, and treats it as a restricted identity function: it cannot create new terms, only propagate them to new sorts, much like how we handle the sort-ordering $\leq$. We stress that this has not solved the sorts-as-predicates problem for two reasons:

- This algorithm treats sorts-as-predicates *globally* rather than locally, which could result in an over-estimation of sufficient model size. For instance, consider the following sentence over a language with two functions $f : B \rightarrow D$ and $g : C \rightarrow E$. $\exists x^A (B(x) \vee C(x))$. By adding two separate coercion functions due to $B(x)$ and $C(x)$ we imply that both branches of the disjunction must be true, and produce the term model $\{c_x, f(c_x), g(c_x)\}$. A more accurate treatment would need to consider multiple term models.

- Since the query language has no support for complex terms, we handle only sort-as-predicate cases where the term appearing is a variable. A full treatment of sorts-as-predicates would need to handle all possible terms.

4.7 Tupling

We have seen that given a query, we can often compute an upper bound on the model sizes at which we must search for solutions. This makes Margrave's answer to many queries sound and complete, but the ceiling is often high, even for simple queries. In this section we present some results that can allow a drastic reduction in sufficient model size for a heavily-used class of queries – one that does not require the full expressive power of OS-EPL.

We first introduce the idea in ordinary first-order logic.

4.7.1 The First-Order Existential Case

Given a query of the form $\exists x_1, ..., \exists x_k \alpha$, where α is quantifier-free, along with various axioms of the form $\forall y_1, ..., \forall y_l \beta$ (where β is also quantifier free), algorithm 2 will return k, the number of existentials, as an upper bound. Yet the query seeks only a *single tuple* that satisfies α, suggesting that there may be potential for improvement.

For example, consider this simple (unsorted) access-control policy:

```
permit(s a r) :- employee(s), read(a), public(r).
permit(s a r) :- manager(s), read(a).
permit(s a r) :- manager(s), write(a), public(r).
permit(s a r) :- employee(s), read(a), manager(m),
                    personal-assistant(m, s).
```

and a query over that policy asking what requests it permits: $\exists s\ \exists a\ \exists r\ Permit(s, a, r)$. Margrave will calculate a bound of 4 for this query, and check for models at sizes

1, 2, 3, and 4. Our goal is to show that a bound of 1 suffices in such situations, although as we show below, we pay a cost: a potential increase in the number of relation symbols in our language.

Given an existential, prenex form sentence $\sigma \equiv \exists x_1 ... \exists x_k \alpha$ (where α is quantifier free using $l \leq k$ free variables) over a constant- and function-free signature Σ (that may use equality), we can construct an equisatisfiable existential sentence, possibly over a new signature, having only one (existential) quantifier.

The intuition here is simple: For each environment η over a model $\mathbb{M}$ there is a *tuple* of elements $(\eta(x_1), ..., \eta(x_k))$ that describes bindings to the variables in $x_1, ..., x_k$. An atomic formula $P(x_i)$ in α asserts that the i^{th} component of that tuple is in the relation $P^{\mathbb{M}}$. The goal is to rewrite σ so that it involves only one existentially quantified variable z representing the entire tuple $(x_1, ..., x_k)$.

If α doesn't contain equality, the rewriting process is straightforward: we need only instrument each atomic formula with an index, describing which component(s) of the tuple it involves. For instance:

Example 11. *$\sigma \equiv \exists x \exists y P(y) \wedge \neg R(x, y)$ would produce $\sigma' \equiv \exists z P_2(z) \wedge \neg R_{1,2}(z)$ over the new signature $\Sigma' = \{P_2, R_{1,2}\}$.*

However, in general α may contain equality, which makes the rewriting slightly more involved. Here we will define some constructions that will prove useful.

Definition 13 (Tupled Signature)**.** *Let $\sigma \equiv \exists x_1 ... \exists x_k \alpha$ (where α is quantifier free) be a sentence over the signature Σ.*

The tupling *of Σ with respect to σ, written as Σ_τ is built as follows:*

- *Σ_τ contains a predicate symbol $P_{i_1,...,i_n}$ of arity 1, (where i_j are integers between 1 and k) for each atom $P(x_{i_1}, ..., x_{i_n})$ that appears in σ.*
- *Σ_τ also contains a predicate symbol $=_{i,j}$ for each $1 \leq i < j \leq k$.*

Definition 14 (Tupled Formula)**.** *Let $\sigma \equiv \exists x_1 ... \exists x_k \alpha$ (where α is quantifier free) be a sentence over the signature Σ.*

The tupling *of σ, written as $\tau(\sigma)$, is a sentence over Σ_τ and is defined to be identical to σ except that:*

- *The prefix of $\tau(\sigma)$ contains a single existential quantifier $\exists z$.*
- *All atoms $P(x_{i_1}, ..., x_{i_n})$ in σ are replaced with $P_{i_1,...,i_n}(z)$ in $\tau(\sigma)$.*
- *Each atom $(x_i = x_j)$ is replaced with $=_{i,j}(z)$ (or $=_{j,i}$ if $j < i$); any occurence of $(x_i = x_i)$ is replaced with the tautology $\top$.*

If the original formula used equality, we must add an axiomatizaiton of equality that makes the new equality predicates behave as we would expect equality to.

Definition 15 (Equality Axioms)**.** *The* equality axioms *for σ, written as $\gamma(\sigma)$, is the sentence over Σ_τ that is the conjunction of the following finite set of universal formulas:*

- *For each $=_{i,j}$ in Σ_τ and each pair of predicate symbols in $P_{i_1,...,i,...,i_n}$ and $P_{i_1,...,j,...,i_n}$ in Σ_τ, add: $\forall z\,(=_{i,j}(z) \implies (P_{i_1,...,i,...,i_n}(z) \iff P_{i_1,...,j,...,i_n}(z)))$*
- *Whenever all of $=_{i,j}$, $=_{j,k}$, and $=_{i,k}$ (with the subscripts in any order) appear in Σ_τ, add: $\forall z\,(=_{i,j}(z) \wedge =_{j,k}(z) \implies =_{i,k}(z))$.*

Note that the second type of axiom above is indeed required. The issue is one of subscript ordering. Suppose only the first type of axiom was added, and that the equality predicates $=_{1,2}$, $=_{2,3}$, and $=_{1,3}$ all appear in Σ_τ. Then the following axioms are produced:

$\forall z\,(=_{1,2}(z) \implies (=_{1,3}(z) \iff =_{2,3}(z)))$

$\forall z\,(=_{2,3}(z) \implies (=_{1,2}(z) \iff =_{1,3}(z)))$

but the crucial third axiom will not be constructed, because there is no predicate named $=_{3,2}$. To correct this, we ignore subscript ordering when forming transitivity axioms above.

Example 12. *Suppose that* $\sigma \equiv \exists x\ \exists y\ R(y,x) \wedge \neg R(x,y) \wedge \neg(x=y)$. *Then:*

$$\begin{aligned} \tau(\sigma) &\equiv \exists z R_{2,1}(z) \wedge \neg R_{1,2}(z) \wedge \neg =_{1,2}(z) \\ \gamma(\sigma) &\equiv =_{1,2}(z) \implies (R_{1,2}(z) \iff R_{2,1}(z)) \end{aligned}$$

.

Note that if σ does not use equality, then the equality predicates and axioms can be disregarded and the following bounds hold:

$|\sigma'| \leq |\sigma|$.

$$|\Sigma'| \leq \sum_{P \in \Sigma \text{ of arity m}} (k^m).$$

If σ uses equality, the bounds increase:

$$|\sigma'| \leq |\sigma| + \tbinom{k}{2}|\Sigma'|$$
$$|\Sigma'| \leq \sum_{P \in \Sigma \text{ of arity m}} (k^m) + \binom{k}{2}.$$

As we will see, these bounds are extremely conservative in practice.

Definition 16 (Product Model)**.** *Let* $\sigma \equiv \exists x_1 ... \exists x_k \alpha$ *(where* α *is quantifier free) be a sentence over the signature* Σ.

Given a model $\mathbb{M}$ *of* Σ*, the* k-ary product model $\mathbb{M}^{\uparrow k}$ *of* $\mathbb{M}$ *is the following model of* Σ_τ.

Let $|\mathbb{M}^{\uparrow k}| = |\mathbb{M}|^k$. *That is,* $|\mathbb{M}^{\uparrow k}|$ *is the set of all k-ary tuples of elements of* $\mathbb{M}$.

For each predicate symbol $P_{i_1,...,i_n} \in \Sigma_\tau$ *and element* $(e_1, ..., e_k) \in |\mathbb{M}^{\uparrow k}|$, $(e_1, ..., e_k) \in P^{\mathbb{M}^{\uparrow k}}_{i_1,...,i_n}$ *if and only if* $(e_{i_1}, ..., e_{i_n}) \in P^{\mathbb{M}}$.

For each $=_{i,j} \in \Sigma_\tau$, $(e_1, ..., e_k) \in =^{\mathbb{M}^{\uparrow k}}_{i,j}$ *if and only if* $e_i = e_j$.

Example 13. $|\mathbb{M}| = \{a, b\}$. $P^{\mathbb{M}} = \{a\}$. $R^{\mathbb{M}} = \{(a, a), (a, b)\}$.
$|\mathbb{M}^{\uparrow 2}| = \{aa, ab, ba, bb\}$. $P_2^{\mathbb{M}^{\uparrow 2}} = \{aa, ba\}$. $R_{12}^{\mathbb{M}^{\uparrow 2}} = \{aa, ab\}$.

Lemma 10. *Let* $\sigma \equiv \exists x_1 ... \exists x_k \alpha$ *(where* α *is quantifier free) be a sentence over the signature* Σ. *If* $\mathbb{M} \models \sigma$, *then* $\mathbb{M}^{\uparrow k} \models \tau(\sigma) \wedge \gamma(\sigma)$.

Proof. Since we have constructed each $=^{\mathbb{M}^{\uparrow k}}_{i,j}$ using true equality in $\mathbb{M}$, $\gamma(\sigma)$ must hold in $\mathbb{M}^{\uparrow k}$. So we only need to show that $\mathbb{M}^{\uparrow k} \models \tau(\sigma)$.

Fix some η such that $\mathbb{M} \models_\eta \alpha$. Let η' be an environment mapping z to $(\eta(x_1), ..., \eta(x_k)) \in \mathbb{M}^{\uparrow k}$. Let α' be the matrix of $\tau(\sigma)$. Now we show by induction that $\mathbb{M}^{\uparrow k} \models_{\eta'} \alpha'$.

- $\alpha' \equiv P_{i_1,...,i_n}(z)$. So $\alpha \equiv P(x_{i_1}, ..., x_{i_n})$. But by our construction of $\mathbb{M}^{\uparrow k}$ and η', $(\eta(x_{i_1}), ..., \eta(x_{i_n})) \in P^{\mathbb{M}}$ if and only if $\eta'(z) \in P_{i_1,...,i_n}$.
- $\alpha' \equiv =_{i,j}(z)$. So $\alpha \equiv (x_i = x_j)$. Again, $\eta'(z) = (\eta(x_1), ..., \eta(x_k))$ and so by our construction of $\mathbb{M}^{\uparrow k}$, $\eta'(z) \in =^{\mathbb{M}^{\uparrow k}}_{i,j}$ iff $\eta(x_i) = \eta(x_j)$.
- $\alpha' \equiv \beta \wedge \gamma$, $\beta \vee \gamma$, or $\neg\beta$: Follows by direct use of our inductive hypothesis on the smaller formulas β and γ.

□

Given a model $\mathbb{M}$ over Σ_τ, we would like to define a construction that "unpacks" tuples into their original form: the reverse of a product model in the natural way. However if $\mathbb{M}$ does not satisfy the axioms in $\gamma(\sigma)$, the "natural" projection of equality

becomes ambiguous. Because of this, we use the equality axioms $\gamma(\sigma)$ to aid us in reversing the product model process.

There is another delicacy: if the size of $\mathbb{M}$ isn't n^k for some n, it isn't immediately apparent how to construct its projection. Nor is it obvious how to "project" predicates. (For example, consider a model with two elements a and b and one predicate symbol P_1 with interpretation $\{a\}$. There is nothing in this model to tell us how large the new model should be, nor how to assign projections from a and b to elements, even if we knew its size.) Fortunately, it turns out that model size 1 suffices.

Definition 17 (Projection Model)**.** *Let $\sigma \equiv \exists x_1 ... \exists x_k \alpha$ (where α is quantifier free) be a sentence over the signature Σ.*

Given some single-element model $\mathbb{M}$ over Σ_τ that satisfies $\gamma(\sigma)$, we define the projection model for $\mathbb{M}$*, written $\mathbb{M}^\downarrow$ as follows.*

Let S be a set of k arbitrary elements $\{e_1, ..., e_k\}$, and let $*$ be the distinct element of $\mathbb{M}$. Now define the equivalence relation R on S such that $R(e_i, e_j) \iff * \in =^{\mathbb{M}}_{i,j}$.

Now let $|\mathbb{M}^\downarrow|$ be the quotient of S with respect to R. We denote the equivalence class of an element e as $[e]$.

For each n-ary predicate P, $([e_{i_1}], ..., [e_{i_n}]) \in P^{\mathbb{M}^\downarrow}$ if and only if $* \in P^{\mathbb{M}}_{j_1,...,j_n}$ for some $1 \leq j_1, ..., j_n \leq k$ and either $i_r = j_r$ or $=^{\mathbb{M}}_{i_r,j_r}$ for each $1 \leq r \leq n$.

This is well-defined since we forced each $=_{i,j}$ to respect predicates in the equality axioms of $\gamma(\sigma)$, and $\mathbb{M} \models \gamma(\sigma)$.

Example 14. *Let Σ and σ be as in the above examples.*

$|\mathbb{M}| = \{*\}$. $P_2^{\mathbb{M}} = \{*\}$. $R_{12}^{\mathbb{M}} = \emptyset$.

Then the projection model for $\mathbb{M}$ is: $|\mathbb{M}^\downarrow| = \{e_1, e_2\}$. $P^{\mathbb{M}^\downarrow} = \{e_2\}$. $R^{\mathbb{M}^\downarrow} = \emptyset$.

Lemma 11. *Let $\sigma \equiv \exists x_1 ... \exists x_k \alpha$ (where α is quantifier free) be a sentence over the signature Σ.*

Given a model $\mathbb{M}$ over Σ_τ, if $||\mathbb{M}|| = 1$ and $\mathbb{M} \models \tau(\sigma) \wedge \gamma(\sigma)$, then $\mathbb{M}^\downarrow \models \sigma$.

Proof. Write the single element of $|\mathbb{M}|$ as $*$

$\mathbb{M} \models_{z \mapsto *} \tau(\sigma) \wedge \gamma(\sigma)$ so of course $\mathbb{M} \models_{z \mapsto *} \tau(\sigma)$. Since $\tau(\sigma)$ is identical to the matrix of σ except for atoms, it suffices to show that $* \in P^{\mathbb{M}}_{i_1,...,i_n}$ if and only if $([e_{i_1}], ..., [e_{i_n}]) \in P^{\mathbb{M}^\downarrow}$. But this is true by our construction of $\mathbb{M}^\downarrow$.

□

Theorem 8. *Given a sentence $\sigma \equiv \exists x_1 ... \exists x_k \alpha$ (where α is quantifier free) over a constant- and function-free signature Σ (that may use equality),*

σ is satisfiable if and only if $\tau(\sigma) \wedge \gamma(\sigma)$ is satisfiable at model size 1.

Proof. Since $\sigma \equiv \exists x_1 ... \exists x_k \alpha$ is prenex existential, a model $\mathbb{I}$ satisfies σ if and only if it has some induced submodel with no more than k elements. Now Lemmas 10 and 11 suffice since $\tau(\sigma)$ has only one existential quantifier. □

Example 15. *We return to the example from the beginning of this section: $\sigma \equiv \exists s \exists a \exists r Permit(s, a, r)$.*

This will be expanded to:

$$\begin{aligned} \exists s \exists a \exists r \quad & (employee(s) \wedge read(a) \wedge public(r)) \vee \\ & (manager(s) \wedge read(a)) \vee \\ & (manager(s) \wedge write(a) \wedge public(r)) \vee \\ & (\exists m \; employee(s) \wedge read(a) \wedge manager(m) \wedge personal\text{-}assistant(m, s)) \end{aligned}$$

.

After prenexing, tupling produces:

$$\begin{aligned}\exists z \quad & (employee_1(z) \wedge read_2(z) \wedge public_3(z)) \vee \\ & (manager_1(z) \wedge read_2(z)) \vee \\ & (manager_1(z) \wedge write_2(z) \wedge public_3(z)) \vee \\ & (employee_1(z) \wedge read_2(z) \wedge manager_4(z) \wedge personal\text{-}assistant_{4,1}(z))\end{aligned}$$

.

Notice that in this particular case, tupling has not increased the number of predicates to consider; we have even reduced a binary relation to a unary one.

After tupling is complete on this query, a model-finder need only consider models of size 1. It would also be possible to immediately apply purely propositional methods (e.g. BDDs) to the tupled formula. While our established bounds on the size of Σ_τ allow for pathological cases where this procedure is actually unhelpful, we have found in practice that tupling nearly always results in a performance gain for our tool (Section 3.5).

4.7.2 The Sorted Case

Margrave uses first-order *order-sorted logic*, and so we must extend the first-order tupling theorem to the order-sorted setting. This both introduces subtleties (as in the case of empty sorts) and grants benefits as we will see. First we redefine the process of tupling in the sorted setting. We allow sorts to appear as predicate symbols in σ, noting that in this restricted case they do not present a problem (Section 4.6).

Definition 18 (Tupled Signature)**.** *Fix an order-sorted vocabulary* Σ *and a sentence* $\sigma \equiv \exists x_1^{S_1} ... \exists x_k^{S_k} \alpha$ *over* Σ *(where* α *is quantifier free).*

The tupling *of* Σ*, denoted* Σ_τ *contains the following sorts:*

- *A unique maximal sort Z.*
- *For each index i, $1 \leq i \leq k$, Σ_τ contains a sort A_i if:*
  - *The atom $A(x_i)$ appears in σ; or*
  - *Sort B_i is in Σ_τ and $B \leq A$ in Σ (in this case $B_i \leq A_i$); or*
  - *$P \in \Sigma_{\langle A_1,...,A_{i-1},A,A_{i+1},...,A_n\rangle}$ and the atom $P(x_{j_1},...,x_i,...,x_{j_n})$ appears in σ.*

If the atom $P(x_{i_1},...,x_{i_n})$ appears in σ, a unary predicate symbol $P_{i_1,...,i_n}$ appears in Σ_τ.

Σ_τ also contains a predicate symbol $=_{i,j}$ for each $1 \leq i < j \leq k$.

Definition 19 (Tupled Formula)**.** *Fix an order-sorted Σ and a sentence $\sigma \equiv \exists x_1^{S_1}...\exists x_k^{S_k}\alpha$ over Σ (where α is quantifier free).*

The tupling *of σ, written $\tau(\sigma)$, is defined to be identical to σ except that:*

- *The prefix of $\tau(\sigma)$ contains a single existential quantifier $\exists z^Z$.*
- *All atoms $P(x_{i_1},...,x_{i_n})$ in σ are replaced with $P_{i_1,...,i_n}(z)$ in $\tau(\sigma)$. This case applies to both predicate and sort symbols.*
- *Each equality atom $(x_i = x_j)$ is replaced with $=_{i,j}$ (z) (or $=_{j,i}$ if $j < i$); any occurence of $(x_i = x_i)$ is replaced with the tautology $\top$.*

Example 16. *Suppose the original vocabulary Σ contains the sorts shown in Figure 4.1 and that the query sentence has the prefix:*

$$\exists ipsrc^{IPAddress}\, \exists ipdest^{IPAddress}\, \exists portsrc^{Port}\, \exists portdest^{Port}$$

and a matrix that involves the atomic formulas $www.wpi.edu(ipdest)$, $www(portdest)$, and $192.168.0.1(ipsrc)$. Then the tupled vocabulary Σ_τ would contain the hierarchy shown in Figure 4.2. Shaded nodes represent sorts explicitly mentioned either in the matrix or in the quantifiers themselves; uncolored nodes represent sorts that were included because they are parents of an explicitly mentioned sort.

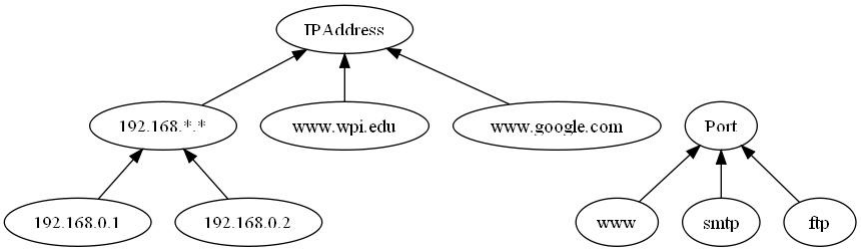


Figure 4.1: Original Hierarchy

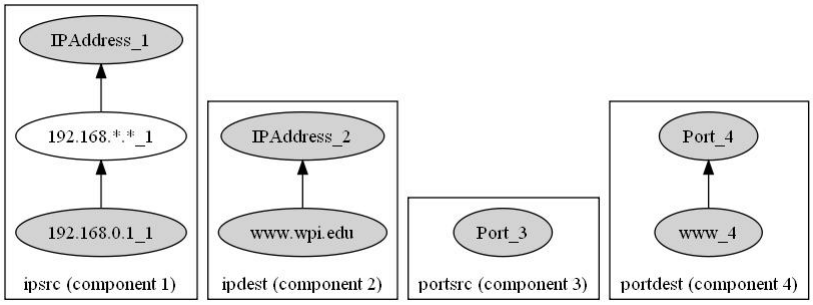


Figure 4.2: Tupled Hierarchy

Definition 20 (Equality Axioms)**.** *Fix an order-sorted vocabulary* Σ *and a sentence* $\sigma \equiv \exists x_1^{S_1} ... \exists x_k^{S_k} \alpha$ *over* Σ *(where* α *is quantifier free).*

The Equality axioms for σ*, denoted* $\gamma(\sigma)$*, are identical to the same in the unsorted case, except quantification is over the maximal sort* Z*.*

The nature of sortedness introduces a difference in quantification between σ and $\tau(\sigma)$. We have introduced a sort symbol Z in Σ_τ that implicitly means "tuples of sort $S_1 \times ... \times S_k$". If we do not make this assumption explicit, we will be unable to prove an analog to Theorem 8 in the sorted case. The following example illustrates the problem.

Example 17. *Let* $\sigma \equiv \exists x^A \neg A(x)$*. Clearly* σ *is unsatisfiable. But* $\tau(\sigma) \equiv \exists z^Z \neg A_1(z)$*, which is satisfiable.*

Definition 21 (Tupling Axioms)**.** *Fix an order-sorted vocabulary* Σ *and a sentence*

$\sigma \equiv \exists x_1^{S_1} ... \exists x_k^{S_k} \alpha$ *over* Σ *(where* α *is quantifier free).*

The tupling axioms *for* σ*, denoted* $\beta(\sigma)$ *consist of the following universally quantified sentence:*

$\forall z^Z S_{1_1}(z) \wedge ... \wedge S_{k_k}(z)$.

Definition 22 (Product Model (Sorted))**.** *Fix an order-sorted* Σ *and a sentence* $\sigma \equiv \exists x_1^{S_1} ... \exists x_k^{S_k} \alpha$ *over* Σ *(where* α *is quantifier free).*

Given a model $\mathbb{M}$ *of* Σ*, we define the* Product Model of $\mathbb{M}$ *with respect to* σ*, written* $\mathbb{M}^{\uparrow S_1,...,S_k}$ *to be a model over* Σ_τ *as follows:*

Let $|\mathbb{M}^{\uparrow S_1,...,S_k}|_Z$ *be* $|\mathbb{M}|_{S_1} \times ... \times |\mathbb{M}|_{S_k}$.

For all other sort symbols A_i *in* Σ_τ*, the element* $(e_1, ..., e_k)$ *is in* $|\mathbb{M}^{\uparrow S_1,...,S_k}|_{A_i}$ *if and only if* $e_i \in |\mathbb{M}|_A$.

For each predicate symbol $P_{i_1,...,i_n} \in \Sigma_{\tau\langle A_1,...,A_n\rangle}$*, each element* $(e_1, ..., e_k)$ *of* $|\mathbb{M}^{\uparrow S_1,...,S_k}|_Z$ *is in* $P^{\mathbb{M}^{\uparrow k}}_{i_1,...,i_n}$ *if and only if* $(e_{i_1}, ..., e_{i_n}) \in P^{\mathbb{M}}$.

For each $=_{i,j} \in \Sigma_\tau$*,* $(e_1, ..., e_k) \in =_{i,j}^{\mathbb{M}^{\uparrow S_1,...,S_k}}$ *if and only if* $e_i = e_j$.

Note that if any of the prefix sorts are empty in $\mathbb{M}$, then $\mathbb{M}^{\uparrow S_1,...,S_k}$ is the empty model for Σ_τ.

Lemma 12. *Fix an order-sorted* Σ *and a sentence* $\sigma \equiv \exists x_1^{S_1} ... \exists x_k^{S_k} \alpha$ *over* Σ *(where* α *is quantifier free).*

If $\mathbb{M} \models \sigma$*, then* $\mathbb{M}^{\uparrow S_1,...,S_k} \models \tau(\sigma) \wedge \gamma(\sigma) \wedge \beta(\sigma)$.

Proof. Again, since we have constructed each $=_{i,j}^{\mathbb{M}^{\uparrow S_1,...,S_k}}$ using true equality in $\mathbb{M}$, $\gamma(\sigma)$ must hold in $\mathbb{M}^{\uparrow S_1,...,S_k}$. We also know that $\mathbb{M}^{\uparrow S_1,...,S_k} \models \beta(\sigma)$ since we constructed the set $|\mathbb{M}^{\uparrow S_1,...,S_k}|_Z$ to contain only properly typed tuples. Thus it only remains to show that $\mathbb{M}^{\uparrow S_1,...,S_k} \models \tau(\sigma)$.

Fix some η such that $\mathbb{M} \models_\eta \alpha$. Let η' be an environment mapping z to $(\eta(x_1), ..., \eta(x_1)) \in |\mathbb{M}^{\uparrow S_1,...,S_k}|_Z$. We know that this element exists since otherwise

the environment η could not exist.

Let α' be the matrix of $\tau(\sigma)$. Now we show by induction that $\mathbb{M}^{\uparrow S_1,...,S_k} \models_{\eta'} \alpha'$. The induction proceeds similarly to the unsorted case, except that there is now a sort-as-predicate case:

$\alpha' \equiv A_i(z)$. So $\alpha \equiv A(x_i)$. Again, by our construction of the product model, $\eta'(z) \in |\mathbb{M}^{\uparrow S_1,...,S_k}|_{A_i}$ if and only if $\eta(x_i) \in |\mathbb{M}|_A$.

□

Definition 23 (Projection Model (Sorted))**.** *Fix an order-sorted Σ and a sentence $\sigma \equiv \exists x_1^{S_1}...\exists x_k^{S_k}\alpha$ over Σ (where α is quantifier free).*

Given a single-element model $\mathbb{M}$ of Σ_τ, that satisfies $\gamma(\sigma)$, we define the Projection of $\mathbb{M}$ *with respect to σ, written $\mathbb{M}^\downarrow$, to be a model over Σ_τ as follows:*

Let S be a set of k arbitrary elements $\{e_1, ..., e_k\}$, and let $*$ be the distinct element of $\mathbb{M}$. Now define the equivalence relation R on S such that $R(e_i, e_j) \iff * \in =_{i,j}^{\mathbb{M}}$. We denote the equivalence class (modulo R) of an element $e \in S$ as $[e]$.

For all each sort symbol A in Σ, and each equivalence class $[e_i]$, the element $[e_i]$ is in $|\mathbb{M}^\downarrow|_A$ if and only if

- $* \in |\mathbb{M}|_{A_i}$*; or*
- *for some j, $* \in =_{i,j}$ and $* \in |\mathbb{M}|_{A_j}$.*

For each n-ary predicate P, $([e_{i_1}], ..., [e_{i_n}]) \in P^{\mathbb{M}^\downarrow}$ if and only if $* \in P^{\mathbb{M}}_{j_1,...,j_n}$ for some $1 \leq j_1, ..., j_n \leq k$ and either $i_r = j_r$ or $=_{i_r,j_r}^{\mathbb{M}}$ for each $1 \leq r \leq n$.

The above constructions are well-defined since $\mathbb{M}$ satisfies $\gamma(\sigma)$.

Lemma 13. *Fix an order-sorted Σ and a sentence $\sigma \equiv \exists x_1^{S_1}...\exists x_k^{S_k}\alpha$ over Σ (where α is quantifier free).*

Given a model $\mathbb{M}$ over Σ_τ, if $||\mathbb{M}|| = 1$ and $\mathbb{M} \models \tau(\sigma) \wedge \gamma(\sigma) \wedge \beta(\sigma)$, then $\mathbb{M}^\downarrow \models \sigma$.

Proof. Write the single element of $|\mathbb{M}|$ as $*$

$\mathbb{M} \models_{z \mapsto *} \tau(\sigma) \wedge \gamma(\sigma) \wedge \beta(\sigma)$ so of course $\mathbb{M} \models_{z \mapsto *} \tau(\sigma)$. Since $\tau(\sigma)$ is identical to σ except for atoms, it suffices to show that:

- The mapping $x_i^{S_i} \mapsto [e_i]$ (for each $1 \leq i \leq k$) is a sorted environment. That is, the mapping respects the sort of each x_i. This is true since $\mathbb{M} \models \beta(\sigma)$.
- $* \in P^{\mathbb{M}}_{i_1,...,i_n}$ if and only if $([e_{i_1}], ..., [e_{i_n}]) \in P^{\mathbb{M}^{\downarrow}}$ and
- $* \in |\mathbb{M}|_{A_i}$ if and only if $[e_i] \in |\mathbb{M}^{\downarrow}|_A$. These are true by our construction of $\mathbb{M}^{\downarrow}$.

□

Lemma 13 is where we use $\beta(\sigma)$. If $\mathbb{M}\neg \models \beta(\sigma)$, we would be unable to construct the environment needed to proceed.

Now we state the existential tupling theorem for order-sorted logic.

Theorem 9. *Given an order-sorted Σ and a sentence $\sigma \equiv \exists x_1^{S_1} ... \exists x_k^{S_k} \alpha$ over Σ (where α is quantifier free),*

σ is satisfiable if and only if $\tau(\sigma) \wedge \gamma(\sigma) \wedge \beta(\sigma)$ is satisfiable at model size 1.

Proof. Since σ is prenex existential, a model $\mathbb{I}$ satisfies σ if and only if it has some induced submodel with no more than k elements. Now Lemmas 12 and 13 suffice since $\tau(\sigma)$ has only one existential quantifier and $\gamma(\sigma) \wedge \beta(\sigma)$ is universal.

□

Note that the presence of empty sorts does not affect the above theorem, but in the sorted setting, prenex normal form may not be attainable in the presence of empty sorts. Here, Margrave must tread carefully. For example, recall our example Permit query above, which is expanded to:

$$\begin{array}{ll} \exists s \exists a \exists r & (employee(s) \wedge read(a) \wedge public(r)) \vee \\ & (manager(s) \wedge read(a)) \vee \\ & (manager(s) \wedge write(a) \wedge public(r)) \vee \\ & (\exists m\ employee(s) \wedge read(a) \wedge manager(m) \wedge personal\text{-}assistant(m, s)) \end{array}$$

.

This sentence contains an existential outside the prefix. If the sort *employee* is allowed to be empty, this query cannot be tupled safely — at least using the current theory.

Since an order-sorted signature with a single sort can simulate ordinary first-order logic, we cannot claim any general performance gains in the sorted case. However we have found that in practice the typing discipline imposed by sorts can result in fewer axioms.

4.7.3 Including Constraints

Even if the user's query is fully existential, it may involve vocabulary constraints that contain implicit universal quantifiers. For instance, a constraint that expresses *A and B are disjoint sorts* requires universal quantification:

$$\forall x^A \forall y^B (x \neq y)$$

.

A constraint expressing *A contains at most one element* does as well:

$$\forall x^A \forall y^A (x = y)$$

.

As does saying that *A is covered by the union of its n subsorts*:

$$\forall x^A(B_1(x) \vee ... \vee B_n(x))$$

.

Though there may be different ways to write these constraints, they cannot be expressed in negation-normal form without universal quantifiers. This fact is easy to see since existential formulas preserve truth in supermodels, but the above formulas do not. Thus such constraints fall outside Theorem 9, since it handles only existential sentences.

It is possible to prove an analog to the existential tupling theorem for universal formulas, but it is more complicated and may cause an exponential blowup in formula size. Since these three constraint types are the only non-functional *universal* constraints that Margrave allows, we opt to show how each of the above may be incorporated into Theorem 9.

Definition 24 (Immediate Subsort)**.** *A sort A where $A \leq B$ is said to be an* immediate subsort *of B if there is no sort C such that $A \leq C \leq B$.*

Definition 25 (Constraint Formula)**.** *Given an order-sorted signature Σ and sort symbols A, B in Σ. A* constraint formula *over Σ is a sentence of the form:*

- $\forall x^A \forall y^B(x \neq y)$*, a* disjointness constraint*; or*
- $\forall x^A \forall y^A(x = y)$*, an* at-most-one *or* lone constraint*; or*
- $\forall x^A(B_1(x) \vee ... \vee B_n(x))$*, a* subsort-exhaustiveness *or* abstract constraint*, provided $B_1, ..., B_n$ are the immediate subsorts of A.*

We now redefine our notion of a tupled signature one last time:

Definition 26. *Fix an order-sorted vocabulary Σ, a sentence $\sigma \equiv \exists x_1^{S_1} ... \exists x_k^{S_k} \alpha$ over Σ (where α is quantifier free), and a set X of constraint formulas over Σ.*

The tupling *of Σ, denoted Σ_τ contains the following sorts:*

- *A unique maximal sort Z.*
- *For each index i, $1 \leq i \leq k$, Σ_τ contains a sort A_i if:*
  - *The atom $A(x_i)$ appears in σ; or*
  - *The sort symbol A is an immediate subsort of B, and B is abstract in X; or*
  - *Sort B_i is in Σ_τ and $B \leq A$ in Σ (in this case $B_i \leq A_i$); or*
  - *$P \in \Sigma_{\langle A_1,...,A_{i-1},A,A_{i+1},...,A_n\rangle}$ and the atom $P(x_{j_1},...,x_i,...,x_{j_n})$ appears in σ.*

If the atom $P(x_{i_1},...,x_{i_n})$ appears in σ, a unary predicate symbol $P_{i_1,...,i_n}$ appears in Σ_τ. Σ_τ also contains a predicate symbol $=_{i,j}$ for each $1 \leq i < j \leq k$.

The above definition makes certain that our notion of projection can function unambiguously in the presence of abstractness constraints.

4.7.3.1 At-most-one

Definition 27 (Tupled At-Most-One Constraint)**.** *Let Σ be a sorted signature and let X be a set of constraint formulas over Σ.*

For each at-most-one constraint formula $\chi \in X$ on sort A in Σ, its tupling*, χ_τ, is the conjunction of the following finite set of formulas over Σ_τ:*

$\{\forall z^Z(A_i(z) \wedge A_j(z)) \implies (=_{i,j}) \mid A_i, A_j\ (i \neq j)\ \textit{appear in}\ \tau(\sigma)\}$.

In the worst case, each at-most-one constraint introduces $\binom{k}{2}$ new formulas.

For this axiom to be complete at all model sizes, we would also need to declare each A_i to be at-most-one in the tupled vocabulary. However, since we will again only be concerned with model size 1 for tupled models, the extra assertions would be pointless.

4.7.3.2 Disjointness

For each $A \perp B$ constraint in the original vocabulary and each index i for which A_i and B_i both appear, we certainly must assert that the sorts A_i and B_i are disjoint in the tupled vocabulary. However, that isn't enough:

Example 18. *Suppose there are sorts A and B, and that $A \perp B$. Let $\sigma \equiv \exists x^A \exists y^B (x = y)$ This sentence is unsatisfiable with respect to the given constraint. The only tupled sorts to appear are A_1 and B_2. Since A_2 does not appear, we cannot directly assert its disjointness from A_1. (Similarly, B_1 does not appear.) This would then result in a satisfiable tupled query.*

Instead, we do the following, which forces a contradiction in the above example since A_1 and B_2 both appear, enforcing $\neg(=_{1,2})$:

Definition 28 (Tupled Disjointness Constraint)**.** *Let Σ be a sorted signature and let X be a set of constraint formulas over Σ.*

For each disjointness formula $\chi \in X$ asserting $A \perp B$ in the original vocabulary, its tupling*, χ_τ, is the conjunction of the following finite sets of formulas over Σ_τ:*

$\{\forall z^Z (A_i(z) \wedge B_j(z)) \implies \neg(=_{i,j}) \mid i \neq j$ A_i and B_j both appear in $\Sigma_\tau\}$; and the new disjointness assertions:

$\{\forall x_i^A \forall y_i^B (x \neq y) \mid 1 \leq i \leq k,$ A_i and B_i both appear in $\Sigma_\tau\}$.

4.7.3.3 Subsort Exhaustiveness

Definition 29 (Tupled Subsort-Exhaustiveness Constraint)**.** *Let Σ be a sorted signature and let X be a set of constraint formulas over Σ.*

For each constraint formula χ that declares a sort A to be exhausted by its immediate subsorts $\{B_1, ..., B_n\}$, its tupling*, χ_τ, is defined to be the conjunction of the following finite set of sentences over Σ_τ:*

$\{\forall x^{A_i}(B_{1_i}(x) \vee ... \vee B_{n_i}) \mid 1 \leq i \leq k\}$.

The intuition for these axioms is: All the immediate subsorts of A_i appear in Σ_τ by definition since A is declared abstract in X. Anything in A_i must belong to one of those subsorts.

Lemma 14. *Fix an order-sorted Σ and a sentence $\sigma \equiv \exists x_1^{S_1}...\exists x_k^{S_k}\alpha$ over Σ (where α is quantifier free). Let χ be a constraint formula.*

If $\mathbb{M} \models \chi$, then $\mathbb{M}^{\uparrow S_1,...,S_k} \models \chi_\tau$.

Proof. We consider each type of constraint formula separately.

- χ is an at-most-one constraint. Then χ_τ is a conjunction of formulas $\forall z^Z(A_i(z) \wedge A_j(z)) \implies =_{i,j}$. All of these must hold in $\mathbb{M}^{\uparrow S_1,...,S_k}$ since there is at most one element e in $|\mathbb{M}|_A$.

- χ is a subsort-exhaustiveness constraint. Then χ_τ is a conjunction of formulas $\forall x^A(B_{1_i}(x) \vee ... \vee B_{n_i}(x))$. These must hold since each element e of $|\mathbb{M}|_A$ is constrained to belong to some $|\mathbb{M}|_{B_j}$.

- χ is a disjointness constraint on A and B. Then clearly for each i, A_i must be disjoint from B_i in $\mathbb{M}^{\uparrow S_1,...,S_k}$. Furthermore, each sentence $\forall z^Z(A_i(z) \wedge B_j(z)) \implies \neg(=_{i,j})$ must hold by construction of $\mathbb{M}^{\uparrow S_1,...,S_k}$.

□

Lemma 15. *Fix an order-sorted Σ and a sentence $\sigma \equiv \exists x_1^{S_1}...\exists x_k^{S_k}\alpha$ over Σ (where α is quantifier free). Let χ_τ be the tupling of a constraint formula.*

If $\mathbb{M} \models \tau(\sigma) \wedge \gamma(\sigma) \wedge \beta(\sigma) \wedge \chi_\tau$, then $\mathbb{M}^{\downarrow} \models \chi$.

Proof. We consider each type of constraint formula separately, showing that the failure of this lemma would lead to a contradiction.

- χ_τ is the tupling of an at-most-one constraint, a conjunction of sentences of the form $\forall z^Z(A_i(z) \wedge A_j(z)) \implies (=_{i,j})$ for all A_i and A_j both appearing in Σ_τ. Now suppose there are two distinct elements $[e_i]$ and $[e_j]$ in $\mathbb{M}^\downarrow$. For all pairs of indices i and j, if both A_i and A_j appear in Σ_τ then χ_τ forces $[e_i] = [e_j]$.

  If it is not the case that A_i appears in Σ_τ then our construction of $\mathbb{M}^\downarrow$ only places $[e_i]$ in $|\mathbb{M}^\downarrow|_A$ if $[e_i] = [e_n]$ for some n with $* \in A_n$. If A_j appears then $=_{n,j}$ holds by χ_τ, and if A_j does not appear then $[e_j] = [e_m]$ for some m with $* \in A_m$, and by χ_τ $[e_m] = [e_n]$.

- χ_τ is the tupling of a subsort-exhaustiveness constraint on some sort A. For each A_i that appears in Σ_τ, χ_τ asserts that $\forall x^{A_i}(B_{1_i}(x) \vee ... \vee B_{n_i})$ holds, where $B_{1_i}, ..., B_{n_i}$ are the immediate subsorts of A_i. Suppose some element $[e_i] \in |\mathbb{M}^\downarrow|_A$. If A_i appears in Σ_τ then the assertions in χ_τ force $[e_i]$ to also belong to an immediate child sort of A_i. If A_i does not appear, then there must be some j for which $* \in A_j$ and $* \in =_{i,j}$. But then the assertions in χ_τ force $[e_j]$ to belong to an immediate child sort of A_j, and $[e_i] = [e_j]$.

- χ_τ is the tupling of a disjointness constraint. We assume there is some $[e_i]$ in both $|\mathbb{M}^\downarrow|_A$ and $|\mathbb{M}^\downarrow|_B$, and show a contradiction. Our construction of $\mathbb{M}^\downarrow$ could have done this in any of four ways:

  - $* \in |\mathbb{M}|_{A_i}$ and $* \in |\mathbb{M}|_{B_i}$. But this cannot happen since χ_τ causes A_i and B_i to be disjoint.
  - $* \in |\mathbb{M}|_{A_i}$, $* \in |\mathbb{M}|_{B_j}$ and $* \in =_{i,j}$. But χ_τ enforces $* \notin =_{i,j}$ and so this cannot happen.
  - By the same argument, the case where $* \in |\mathbb{M}|_{B_i}$, $* \in |\mathbb{M}|_{A_j}$ and $* \in =_{i,j}$ cannot occur either.

– $* \in |\mathbb{M}|_{A_n}$, $* \in |\mathbb{M}|_{B_m}$, $* \in =_{i,m}$ and $* \in =_{i,n}$. But then by the equality axioms $* \in =_{n,m}$ which is forbidden by χ_τ.

So we have a contradiction.

□

Now we state our final theorem:

Theorem 10. *Given an order-sorted* Σ *and a sentence* $\sigma \equiv \exists x_1^{S_1} ... \exists x_k^{S_k} \alpha$ *over* Σ *(where* α *is quantifier free) and a set* X *of constraint formulas,*

$\{\sigma\} \cup X$ *is satisfiable if and only if* $\{\tau(\sigma) \wedge \gamma(\sigma) \wedge \beta(\sigma)\} \cup X_\tau$ *is satisfiable at model size* 1.

Proof. For the only-if direction, Lemma 12 and Lemma 14 together suffice. For the if direction, Lemmas 13 and 15 suffice since $\tau(\sigma)$ has only one existential quantifier. □

4.7.4 Finishing The Example

Now we return to the example at the beginning of this section and treat its sorted nature along with its vocabulary contraints. The original vocabulary has the following sorts:

Subject	*Action*	*Resource*
$Employee \subseteq Subject$	$Read \subseteq Action$	$Public \subseteq Resource$
$Manager \subseteq Subject$	$Write \subseteq Action$	

along with one binary predicate: $Personal\text{-}assistant \subseteq (Subject \times Subject)$.

The tupled vocabulary will contain the following:

$Subject_1$ and $Subject_4$	$Action_2$	$Resource_3$
$Employee_1 \subseteq Subject_1$	$Read_2 \subseteq Action_2$	$Public_3 \subseteq Resource_3$
$Manager_1 \subseteq Subject_1$	$Write_2 \subseteq Action_2$	
$Manager_4 \subseteq Subject_4$		

along with one *unary* predicate: $Personal\text{-}assistant_{4,1}$.

Now, suppose that our example policy's vocabulary also has the following constraints:

```
read and write are disjoint
read contains at most one element
write contains at most one element
```

The two at-most-one constraints do not induce any axioms, since *Read* and $Write$ have only one index (2). The disjointness constraint forces $Read_2 \perp Write_2$ in the new vocabulary, but no axioms.

4.8 Summary

We have identified a decidable class of order-sorted logic, OS-EPL. If a Margrave query coorisponds to a sentence in OS-EPL, Margrave renders sound, complete, and exhaustive results. If a query does not fall into OS-EPL, the user must provide a model-size bound, and Margrave's results are only complete up to that bound. We have given efficient algorithms for deciding membership in this class and generating a tight – for signatures without overloading, as in Margrave – bound on sufficient model size. These algorithms have been implemented in Margrave.

The theoretical framework of tupling has also been implemented in Margrave, and has proven beneficial for queries with large existential prefixes. We see this theory as a way to import certain insights from predicate logic (namely parts of Her-

brand's Theorem; Section 4.4.4) to model-finding tools like Kodkod which propositionalize first-order formulas without taking their structure into account. This insight is due to the fact that the tupling process implicitly uses the fact that given the sentence $\exists x \exists y \ A(x) \wedge B(y)$, which Skolemizes to $A(c_x) \wedge B(c_y)$, we need not consider whether $A(c_y)$ or $B(c_x)$ hold in a model.

Chapter 5

Evaluation

We evaluate Margrave in three different real-world scenarios:

1. an existing XACML policy for CONTINUE, a publically available conference manager;
2. an isolated firewall policy for a real-world enterprise network; and
3. a request for help configuring a Cisco router found on a large networking forum.

An aside on performance benchmarks Margrave's core is written in Java, which means that performance depends on the behavior of the underlying virtual machine: its class loader, garbage collector, JIT compiler, etc. Because of this behavior, we have attempted to measure Margrave's *steady state* performance on all benchmarks in this thesis. We do not report the time that the virtual machine spends "warming up" code.

Moreover, we report averages taken from a pool of trials; the sample size taken varies by test. While this is important in any benchmarking adventure, it is doubly so in a language like Java, where elements of the VM (e.g. the garbage collector)

could intervene at any moment. When measuring the time needed to load a policy from disk, we load different copies of that policy so that the disk cache does not bias the numbers. We also cleared our canonical formula map between attempts.

All performance tests ran on an Intel Core Duo E7200 at 2.53 Ghz with 2 GB of RAM, running Windows XP Home.

5.1 CONTINUE (XACML 1.1)

We first evaluate Margrave on the same XACML policies found in the original change-impact analysis paper by Fisler, *et al.* [FKMT05]. Most notable is the sizable policy set (25 sub-policies, 86 base rules) for CONTINUE, a publically available conference management system.

Loading the CONTINUE policy and running change-impact queries consumed 7 MB of heap space in the Java VM. On average the policy took 106ms to load.[1] Simple verification queries executed in roughly 108ms. With tupling enabled, the same queries executed in around 30ms, half of which was spent tupling the query prior to solving it. These results are comparable to those in the 2005 tool, though the prior work retains a small advantage. Change-impact queries involving the CONTINUE policy and a mutant policy took 275ms on average without tupling, and 79ms with tupling, which is significantly slower than in the 2005 work.

We believe that this difference is due to choice of data structure. The BDDs used in the original 2005 tool will simplify themselves even as they are being built, yet the Kodkod abstract syntax objects used in this thesis do not self-simplify. The change-impact queries that we tested above produced formula trees of just over 10000 nodes (yet only 450 actual formula objects). It is likely that a better data

[1] Each benchmark in this section was the average of 50 samples.

structure or more advanced approach to simplification could substantially reduce this difference.

5.2 A Large Firewall Policy

We have obtained a large Cisco IOS configuration file, containing ACLs for 6 interfaces totalling 1108 rules, which runs on a Cisco router in an enterprise network. This policy is our primary performance benchmark.

Loading the policy took an average of 14 seconds, and consumed roughly 100 MB of memory. Of that, 60 MB was JVM heap and 19 MB was non-heap.

To demonstrate the usefulness of our REPL interface, we wrote a script that detects and explains impotent rules (rules that can never take effect) and overshadowed rules (impotent rules that are "shadowed" by a higher-priority rule with the opposite decision).

The script first executes one query for each rule, asking if it will ever fire. We found that this takes 4.7 minutes on average (for 1108 sub-queries), and the results were surprising: 900 of the 1108 rules in this policy never take effect.

The second portion of our script discovers, for each impotent rule R, which higher-priority rules cover it. It also highlights which of these have a different decision from R. This is useful information since an impotent rule is only inefficient, not erroneous, otherwise. This script segment executed in 114 minutes (comprising several thousand subqueries), and discovered that 274 of the impotent rules were at least partially overshadowed. Memory consumption remained roughly constant at 100 MB throughout the script's execution.

This script demonstrates Margrave's extreme flexibility: users can write such a script to answer any such question that they might have about a policy's behavior,

though the performance of such scripts could be improved: a great deal of setup computation is repeated when executing large numbers of queries.

We also tested the performance of change-impact analysis queries for this firewall policy. To do this, we created a mutant version of the original policy and instructed Margrave to characterize the changes. With tupling, Margrave started to return differences within 1328ms, including query creation, of which 281ms was spent on tupling.

5.3 Help! My Router Isn't working! (IOS, Routing)

We have also sought out real-world examples on Internet help forums: *Ars Technica*[2], *Experts Exchange*[3], and *Networking-forum.com*[4]. This section presents one of those examples which has been treated by our collaborator Christopher Barratt.

This example illustrates Margrave's flexibility: we asked queries that involve multiple ACLs, NAT and routing policies and their interaction. With tupling, all of these queries ran in under 100ms, and the memory footprint to (including loading all component policies) was 40 MB (9.5MB JVM heap, 17.2MB JVM non-heap).

5.3.1 The Cry For Help

In this example[5], an administrator is trying to create two logical networks: one "primary" (consisting of 10.232.0.0/22 and 10.232.100.0/22) and one "secondary" (consisting of 10.232.4.0/22 and 10.232.104.0/22). Neither network should have

[2] `http://episteme.arstechnica.com/eve/forums`

[3] `http://www.experts-exchange.com/`

[4] `http://www.networking-forum.com`

[5] The original help request can be found at: `http://www.experts-exchange.com/Hardware/Networking_Hardware/Q_24113014.html`.

access to the other, but both networks should have access to the Internet—the primary via 10.232.0.15, the secondary via 10.232.4.10:

```
interface GigabitEthernet0/0
description $ETH-LAN$$ETH-SW-LAUNCH$$INTF-INFO-GE 0/0$
ip address 10.232.4.1 255.255.252.0 secondary
ip address 10.232.0.1 255.255.252.0
ip access-group 101 in
ip policy route-map internet
duplex auto
speed auto
!
interface GigabitEthernet0/1
ip address 10.232.8.1 255.255.252.0
duplex auto
speed auto
!
interface Serial0/3/0:0
ip address 10.254.1.129 255.255.255.252
ip access-group 102 out
encapsulation ppp
!
ip route 10.232.100.0 255.255.252.0 10.254.1.130
ip route 10.232.104.0 255.255.252.0 10.254.1.130
!
access-list 102 deny ip 10.232.0.0 0.0.3.255 10.232.104.0 0.0.3.255
access-list 102 deny ip 10.232.4.0 0.0.3.255 10.232.100.0 0.0.3.255
access-list 102 permit ip any any
access-list 101 deny ip 10.232.0.0 0.0.3.255 10.232.4.0 0.0.3.255
access-list 101 deny ip 10.232.4.0 0.0.3.255 10.232.0.0 0.0.3.255
access-list 101 permit ip any any
!
access-list 10 permit 10.232.0.0 0.0.3.255
access-list 10 permit 10.232.100.0 0.0.3.255
access-list 20 permit 10.232.4.0 0.0.3.255
access-list 20 permit 10.232.104.0 0.0.3.255
!
route-map internet permit 10
match ip address 10
set ip next-hop 10.232.0.15
!
route-map internet permit 20
```

```
match ip address 20
set ip next-hop 10.232.4.10
!
end
```

A diagram of the network topology can be found in Figure 5.1 [6].

The configuration here is given for the TAS router.

5.3.2 Finding a Solution

Margrave confirms that network 10.232.0.0/22 cannot reach 10.232.100.0/22 via the serial link:

```
(and
(GigabitEthernet0/0 entry-interface)
(ip-10-232-0-0/ip-255-255-252-0 src-addr-in)
(ip-10-232-100-0/ip-255-255-252-0 dest-addr-in)
(Serial0/3/0:0 exit-interface)

#;> (is-query-satisfiable? routed-packets)
#f
```

Margrave also confirms that network 10.232.4.0/22 cannot reach its Internet gateway 10.232.4.10 via the appropriate Ethernet link:

```
(and
(GigabitEthernet0/0 entry-interface)
(ip-10-232-4-0/ip-255-255-252-0 src-addr-in)
(ip-0-0-0-0/ip-0-0-0-0-other dest-addr-in)
(ip-10-232-4-10 next-hop)
(GigabitEthernet0/1 exit-interface))

#;> (is-query-satisfiable? routed-packets)
#f
```

Relaxation of the first query demonstrates the problem:

[6] This diagram was included as part of the forum help post, and was presumably created by the post's author.

```
(and
(GigabitEthernet0/0 entry-interface)
(ip-10-232-0-0/ip-255-255-252-0 src-addr-in)
(ip-10-232-100-0/ip-255-255-252-0 dest-addr-in)

>#;> (pretty-print-results? routed-packets)
------------------------------------------------------
*** SOLUTION: Size = 7.
$protocol: prot-tcp
$src-addr-in=$src-addr_=$src-addr-out=$next-hop: ip-10-232-0-15
$message: icmp-n/a
$length: len-other
$entry-interface=$exit-interface: gigabitethernet0/0
$src-port-in=$src-port_=$src-port-out=$dest-port-in=$dest-port_=$dest-port-out:
port-n/a
$dest-addr-in=$dest-addr_=$dest-addr-out: ip-10-232-100-0/ip-255-255-252-0
```

All packets from 10.232.0.0/22 are being directed to the Internet gateway for the primary network, rather than only those not intended for 10.232.100.0/22. A simple change in the routing policy fixes this issue–the insertion of the keyword default:

```
.
.
.
route-map internet permit 10
match ip address 10
set ip default next-hop 10.232.0.15
!
route-map internet permit 20
match ip address 20
set ip default next-hop 10.232.4.10
!
.
.
.
```

This change ensures that packets are routed to the Internet only as a last resort (i.e., when static destination-based routing fails). Margrave confirms the change with the new policy:

```
(and
(GigabitEthernet0/0 entry-interface)
(ip-10-232-0-0/ip-255-255-252-0 src-addr-in)
(ip-10-232-100-0/ip-255-255-252-0 dest-addr-in)
(Serial0/3/0:0 exit-interface)

#;> (is-query-satisfiable? routed-packets)
#t

(and
(GigabitEthernet0/0 entry-interface)
(ip-10-232-0-0/ip-255-255-252-0 src-addr-in)
(ip-0-0-0-0/ip-0-0-0-0-other dest-addr-in)
(ip-10-232-0-15 next-hop)
(GigabitEthernet0/0 exit-interface)

#;> (is-query-satisfiable? routed-packets)
#t
```

We also confirm that this change does not suddenly enable the primary sub-network 10.232.0.0/22 to reach the secondary sub-network 10.232.4.0/22:

```
(and
(GigabitEthernet0/0 entry-interface)
(ip-10-232-0-0/ip-255-255-252-0 src-addr-in)
(ip-10-232-4-0/ip-255-255-252-0 dest-addr-in)

#;> (is-query-satisfiable? routed-packets)
#f
```

Examination of whether the secondary network 10.232.4.0/22 can now reach its Internet gateway 10.232.4.10 tells us that we still have more work to do:

```
(and
(GigabitEthernet0/0 entry-interface)
(ip-10-232-4-0/ip-255-255-252-0 src-addr-in)
(ip-0-0-0-0/ip-0-0-0-0-other dest-addr-in)
(ip-10-232-4-10 next-hop)

#;> (pretty-print-results? routed-packets)
#f
```

A close examination of the network diagram reveals a fundamental problem: The gateway 10.232.4.10 should be "on" the same network as the GigabitEthernet0/1 interface, which has an address of 10.232.8.1/22. Relaxation of the above query shows the correct selection of the next-hop gateway, but that the network topology will need to be fixed before the router can successfully forward packets:

```
(and
(GigabitEthernet0/0 entry-interface)
(ip-10-232-4-0/ip-255-255-252-0-other src-addr-in)
(ip-0-0-0-0/ip-0-0-0-0-other dest-addr-in)
(ip-10-232-4-10 next-hop)

#;> (pretty-print-results routed-packets)

*** SOLUTION: Size = 8.
$src-addr-in=$src-addr_=$src-addr-out: ip-10-232-4-0/ip-255-255-252-0-other
$protocol: prot-tcp
$next-hop: ip-10-232-4-10
$message: icmp-echo-reply
$length: len-other
$entry-interface=$exit-interface: gigabitethernet0/0
$src-port-in=$src-port_=$src-port-out=$dest-port-in=$dest-port_=$dest-port-out:
port-n/a
$dest-addr-in=$dest-addr_=$dest-addr-out: ip-0-0-0-0/ip-0-0-0-0-other
```

Changing the address of either the GigabitEthernet0/1 interface or of the next-hop router (10.232.4.10) so that the two share a common network address can remedy this problem.

5.4 Summary

We have shown that Margrave can be applied to a wide range of existing policy languages, and can provide useful feedback to policy authors even in situations where multiple policy types interact. Additionally, Margrave scales to single large-

sized policies, although we expect that exhancements will be needed model entire enterprise networks.

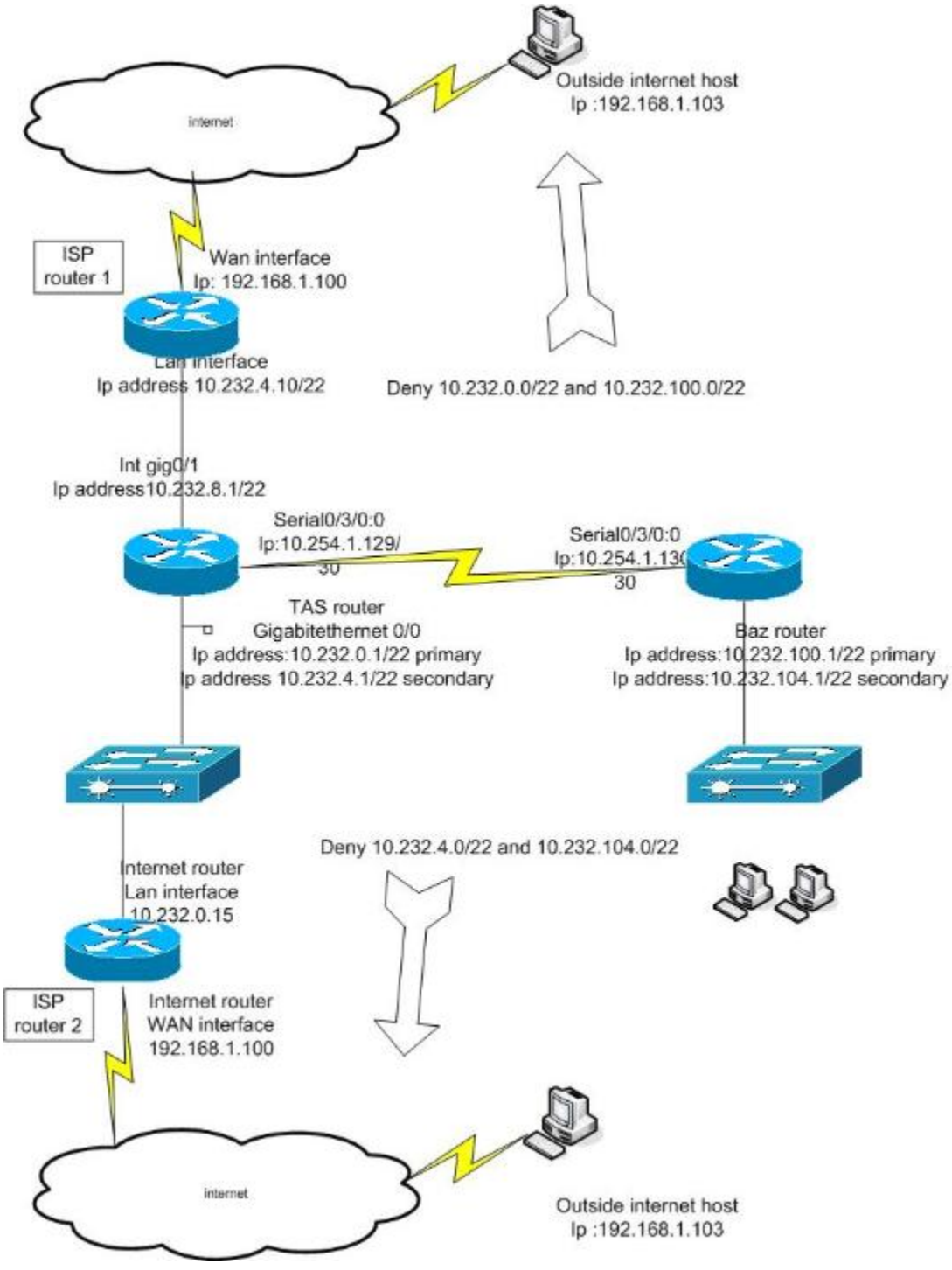


Figure 5.1: The cry for help: network topology

Chapter 6

Related Work

6.1 Policy Analysis

Oppenheimer, *et al.* [OGP03] examine three different internet systems and survey these services' failures over a period of several months. They find that operator error is the leading cause of failure for 2 of the 3 services, and that these errors are predominately configuration problems arising when *changes were made* to the configuration. The work also discusses ways of avoiding and mitigating these failures. Interestingly, it notes that many configuration problems are likely to avoid detection by conventional testing.

Wool [Woo04] studies the prevalence of 12 common configuration errors in firewalls. He shows that larger firewall rule-sets are plagued by a much higher ratio of errors-to-rules than smaller ones, concluding that complex rule sets are simply too difficult for a human administrator to manage unaided.

Eronen and Zitting [EZ01] perform policy analysis on Cisco router ACLs using an existing Constraint Logic Programming framework, which is based on Prolog. Using CLP allows for easy integration of expert knowledge. Users are allowed to

define their own custom IDB predicates as they could in Prolog, which results in a great deal of flexibility. This work is similar to ours in spirit, but is limited to firewall ACLs and does not support NAT or routing information.

Mayer, Wool and Ziskind [MWZ00, MWZ05] and Wool [Woo01] present a tool for discovering what sorts of packets are allowed through a network – possibly containing many routers. Given a query of the form $(Source, Destination, Service)$ and an optional true source (to detect spoofing vulnerability) the tool produces a set of sub-queries describing the packets that are allowed to reach their destination. NAT rules are taken into account. This tool covers only a small subset of what Margrave is capable of, but when the tools overlap, it is much more efficient than Margrave. It is being marketed as a commercial firewall analysis tool.[1]

Liu [Liu07] gives algorithms for performing change-impact analysis on firewall policies. These algorithms are efficient yet only work for atomic changes to a single rule base (adding or deleting a single rule, swapping a pair of rules, or editing a single rule) and do not consider NAT.

Liu and Gouda [LG04] and Gouda and Liu [GL07] introduce a data structure called Firewall Decision Diagrams (FDDs) and use them to assist in the design process for firewall rule sets. Liu and Gouda [LGar] then use FDDs to answer SQL-like queries about a firewall policy. FDDs are an efficient variant of BDDs, and apply only to the firewall packet filtering domain. The authors give algorithms for taking the conjunction, disjunction, and difference of FDDs, but the work does not consider NAT or routing information. This approach accomplishes only some of what Margrave does, but vastly outperforms our tool in those cases.

Marmorstein and Kearns [MK05b, MK05a] introduce a tool, ITVal, that uses Multi-way Decision Diagrams (MDDs) to execute SQL-like queries on firewall poli-

[1] `http://www.algosec.com/en/index.php`

cies. They then expand ITVal to support NAT along with reasoning about hierarchical (serially composed) firewall policies. Their work is also less general than our approach, but more efficient.

Marmorstein and Kearns [MK06] also give a method for generating an equivalence relation on hosts with respect to a policy: two hosts are related if identical (modulo source address) packets from both are treated identically by the firewall. This equivalence class structure can then be used to detect policy anomalies and help administrators understand the policies they write. Since no query is required to generate a set of host classes, this work bears some resemblance to change-impact analysis.

Yuan, *et al.* [YMS+06] use Binary Decision Diagrams (BDDs) to model and analyze large networks of firewall ACLs. The tool ("Fireman") is capable of quickly detecting whitelist and blacklist violations as well as conflicting, redundant, and correlated rules — even between different ACLs. Fireman examines all possible routes through a network at once, but does not consider NAT. Margrave is capable of performing similar checks, although Fireman is 2 to 3 orders of magnitude faster at present. However, Margrave is far more general than Fireman.

Oliveira, *et al.* [OLK09] extend the Fireman [YMS+06] tool to include NAT and take advantage of routing table information. This tool ("Prometheus") can also provide information on which ACL rules are responsible for a misconfiguration. Prometheus has been tested on and is currently undergoing further development by a major European internet service provider.

Verma and Prakash [VP05] implemented FACE, a tool with a dual purpose: automatic configuration of distributed firewalls and analysis of existing distributed firewalls. Our work does not consider generation of configuration files. The queries that FACE makes available are similar to Margrave's in the firewall setting, and

it would be possible to run some change-impact queries in FACE. FACE uses a depth-first search approach to propagate queries through a network; in this way it is similar to the approach of Mayer, Ziskind, and Wool [MWZ05]. FACE is limited to iptables firewalls, although it could be extended. It reasons about large network topologies in a more efficient way than we do. Margrave's API is more general and supports richer policies.

Hughes and Bultan [HB08] use SAT solving to verify properties of XACML policies. Their handling of XACML is more sophisticated than ours; for instance, they allow request contexts to contain set variables. We have opted to be more general and provide an API that can support numerous languages. In order to support policies that involve variable domains of unbounded size, their tool requires a bound on all domain sizes, up to which it performs verification. In this sense our works are similar: both require bounds on model sizes and both use SAT solving, although our use of SAT is mediated by the Kodkod model finder. Margrave possesses the ability to automatically detect many situations where it is possible to bound model sizes, whereas their tool must presumably rely on domain-specific knowledge.

6.2 Order-Sorted Logic

The decidability of the satisfiability problem for the $\exists\forall$ class in pure logic is a classical result of Schönfinkel and Bernays [BS28] in the absence of equality, extended by Ramsey [Ram30] to allow equality. The problem is known to be EXPTIME-complete [Lew80].

The monograph by Börger, Grädel, and Gurevich [BGG97] is an comprehensive treatment of decidability for classical unsorted logic. Herbrand's Theorem [Her30] is the basis for automated deduction and propositional methods.

Enderton [End72] is a textbook treatment of many-sorted logic classically considered. In this setting sorts are assumed non-empty and pairwise disjoint. Strictly speaking, what is assumed is that there is no cross-sort equality; this implies that any model is elementarily equivalent to one with disjoint sorts. An example of the usefulness of multiple sorts in pure logic is Feferman's work [Fef74] on interpolation theorems.

Goguen and Meseguer did seminal work [GM92] on order-sorted algebra; Goguen and Diaconescu [GD94] present a good survey of the field through 1994. Order sorted predicate logic was first considered by Arnold Oberschelp [Obe89]; Walther [Wal88] explores many-sorted unification in the context of orderings on sorts, Weibel [Wei97] extends this work to the order-sorted case.

Harrison was one of the first to observe that many-sortedness can not only yield efficiencies in deduction but can also support new decidability results. In unpublished notes [Harpt] he presents some examples of this phenomenon, and suggests searching for typed analogs of classical decidability classes, as we have done here.

Fontaine and Gribomont [FG03], working in "flat" many-sorted logic (*i.e.*, without subsorting) prove that if there are no functions having result sort A and σ is a universal sentence then σ has a model if and only if it has a model in which the size of A is bounded by the number of constants of sort A. This result is used to eliminate quantifiers in certain verification conditions. This theorem has application even when not all sorts are finite and can be used in a setting where some functions and predicates are interpreted. As observed in Remark 2, the algorithm in Theorem 7 can be used to apply their techniques to a wider class of formulas than they address.

Claessen and Sorensson [CS03b] have integrated a *sort inference* algorithm into the Paradox model-finder that deduces sort information for unsorted problems and,

under certain conditions, can bound the size of domains for certain sorts and improve the performance of the instantiation procedure. Order-sorting is not used, and there are restrictions on the use of equality.

Momtahan [Mom05] defines a fragment of the Alloy kernel language and proves a result computing a refutationally-complete upper bound on the size of a single sort (as a function of the user-provided bounds on the other sorts). The conditions defining this fragment are not directly comparable to ours, but in some respects constrain the sentences rather severely. For example existential quantification in the scope of more than one universal quantifier are usually not allowed.

Abadi *et al.* [ARS10] identify, as we do, a decidable fragment of sorted logic that is decidable by virtue of having a finite Herbrand universe. Although they target Alloy in their examples they work in a many-sorted logic without subsorts or empty sorts; their condition for decidability is the existence of a "stratification" of the function vocabulary; they do not provide algorithms for checking the stratification condition or computing size bounds on the models.

Ge and de Moura [GM09] present a powerful method for deciding satisfiability modulo theories with an instantiation-based theorem prover. Given a universal (Skolemized) sentence σ they construct a system of set constraints whose least solution constitutes a set of ground terms sufficient for instantiation; satisfiability is thus decidable for the set of sentences for which this solution-set is finite (in the many-sorted setting this subsumes the Abadi *et al.* class). They do not treat empty sorts nor subsorting. They can treat certain sentences that fall outside our OS-EPL class; detection of whether a given sentence falls into their decidable class seems to require solving the associated set-constraints, as compared to our linear-time algorithm. Generally speaking they do detailed fine-grained analysis of individual sentences; we have focused on an easily recognized class of sentences.

The problem of efficiently deciding satisfiability in the EPL class is an active area of research. Jereslow [Jer88] described a "partial instantiation" approach to first-order theorem proving in the EPL fragment, constructing a sequence of propositional instantiations instead of working with the full set of possibilities from the outset. Work by Hooker *et al.* [HRCS02] builds directly on Jereslow's approach (see also many references there). Recent alternatives approaches include [dMB08b] and [LS04]. De Moura and Bjørner [DMB08a] have developed the SMT constraint solver Z3. SMT enriches propositional satisfiability by adding equality reasoning, arithmetic, bit-vectors, arrays, and some quantification. Z3 is used in software verification and analysis applications. De Moura and Bjørner [dMB08b]; and Piskac and de Moura and Bjørner [PdMB08], introduce a DPLL-based decision procedure for the EPL class; this has been implemented as part of Z3.

Our work is complementary to these efforts in that it identifies an extended class of sentences to which contemporary techniques can hopefully be applied.

Chapter 7

Conclusion

7.1 Conclusion

This thesis presents Margrave, a tool for analyzing policies that have a notion of predicates and quantification in their rules. We have gone beyond simple access-control, showing Margrave's applicability to enterprise firewall policies and networks, and detailed the logical methods that we use to achieve this.

Perspective A major long-term goal for this project is to provide a core engine that can be used to analyze policies across multiple problem domains. Policies in different domains may look different; we provide an environment where users can reason about interactions between multiple policy domains. Although we have made progress in this thesis, some issues remain for future work.

Once a sufficient number of the remaining issues have been addressed, we will make the tool publically available at `www.margrave-tool.org`. Even now, a pre-release version of the tool is available via the contact information given on the Margrave website.

7.2 Future Work

Interface We acknowledge that the policy (.p), vocabulary (.v), and query languages are not ideal; users (the author included) have found them frustrating. They were created to give us a platform on which to test our core engine. Now that the tool's applicability has been tested, we are addressing these user interface issues.

While Margrave supports reasoning about requests that are modified during evaluation (such as in our Section 2.1 example) it still remains for the user to write queries in a way that accounts for the modification. This is a major usability problem. If the process of writing a query is so technical that it nearly reveals the solution without invoking the tool, then the tool is not useful. The same problem applies to IDB output: users must specify which IDBs they are concerned with. Tupling with IDB output introduces even more complexity for the user, since he or she must say which tuple of variables they care about.

Presenting Solutions As mentioned in Section 3.1, when asked to give *all* solutions Margrave lists all first-order models of the user's query up to a fixed size. Our model-finder, Kodkod, has implemented techniques to remove isomorphic models but we are still often left with exponentially more models than we actually need. While tupling can mitigate the problem the basic issue remains. We are currently investigating techniques that may allow us to show only models that are *representative* for the query, which would prevent the exponential blowup in models shown.

Another concern, twin to the blowup in output length, is the question of how best to *present* the output to the user. A list of first-order models is fine as a proof-of-concept, but it is difficult for most users to understand. New ways of letting users explore the space of their solutions is an exciting direction for future work.

Performance There is room for improving the performance of Margrave, whether through further optimizations of our current method or adoption of new techniques. As we saw in Section 5.2, executing scripts that are comprised of many queries may take more time than most users would be willing to wait.

Policy features The current Margrave tool does not support reasoning about arbitrary-length state changes. This is in part a limitation of first-order logic, which is also known to be unable to express reachability. We can reason about how a packet changes as it passes through a fixed-size network, and we can represent the state table of a firewall, but we do not currently address truly *temporal* policies and queries. This is an interesting avenue of future work, but it involves carefully spying out the land before commiting our forces.

For many of the same reasons, Margrave has no true support for delegation of unbounded length. While Kodkod provides a transitive closure operator, we cannot provide any bounds on sufficient model size when it is used.

Theory Our work in order-sorted logic leaves the problem of sorts-as-predicates only partially solved. While our implementation addresses the problem as much as we needed to for this work (Section 4.6), much is left undone.

Bibliography

[ARS10] Aharon Abadi, Alexander Rabinovich, and Mooly Sagiv. Decidable fragments of many-sorted logic. *Journal of Symbolic Computation*, 45(2):153 – 172, 2010. Automated Deduction: Decidability, Complexity, Tractability.

[BGG97] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

[BS28] Paul Bernays and Moses Schönfinkel. Zum entscheidungsproblem der mathematischen Logik. *Mathematische Annalen*, 99:342–372, 1928.

[CK73] Chen Chung Chang and Jerome Keisler. *Model Theory*. Number 73 in Studies in Logic and the Foundations of Mathematics. North-Holland, 1973. Third edition, 1990.

[CS03a] K. Claessen and N. Sorensson. New techniques that improve MACE-style finite model finding. In *Proceedings of the CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications*. Citeseer, 2003.

[CS03b] K. Claessen and N. Sorensson. New techniques that improve MACE-style finite model finding. In *Proceedings of the CADE-19 Workshop on Model Computation*, 2003.

[DMB08a] L. De Moura and N. Bjorner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, page 337. Springer, 2008.

[dMB08b] Leonardo Mendonça de Moura and Nikolaj Bjørner. Deciding Effectively Propositional Logic Using DPLL and Substitution Sets. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 410–425. Springer, 2008.

[Dum] DumbLittleMan. Death by Google Calendar: How I Identified you to rob you (Access Date: November 24, 2008).

http://www.dumblittleman.com/2006/09/how-to-get-robbed-killed-or-stalked-by.html.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[EZ01] Pasi Eronen and Jukka Zitting. An expert system for analyzing firewall rules. In *Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, pages 100–107, 2001.

[Fef74] S. Feferman. Many-Sorted Interpolation Theorems and Applications. In *Proceedings of the Tarski Symposium, AMS Proc. Symp. in Pure Math*, volume 25, pages 205–223, 1974.

[FG03] Pascal Fontaine and E. Pascal Gribomont. Decidability of invariant validation for parameterized systems. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 2619 of *Lecture Notes in Computer Science*, pages 97–112. Springer-Verlag, 2003.

[FKMT05] Kathi Fisler, Shriram Krishnamurthi, Leo Meyerovich, and Michael Tschantz. Verification and change impact analysis of access-control policies. In *International Conference on Software Engineering (ICSE)*, 2005.

[GD94] Joseph A. Goguen and Razvan Diaconescu. An oxford survey of order sorted algebra. *Mathematical Structures in Computer Science*, 4(3):363–392, 1994.

[GL07] Mohamed G. Gouda and Alex X. Liu. Structured firewall design. *Journal of Computer Networks (Elsevier)*, 51, no. 4:1106–1120, 2007.

[GM92] Joseph A. Goguen and José Meseguer. Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. *Theor. Comput. Sci.*, 105(2):217–273, 1992.

[GM09] Yeting Ge and Leonardo Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *CAV '09: Proceedings of the 21st International Conference on Computer Aided Verification*, pages 306–320, Berlin, Heidelberg, 2009. Springer-Verlag.

[Harpt] John Harrison. Exploiting sorts in expansion-based proof procedures, Unpublished manuscript. `http://www.cl.cam.ac.uk/~jrh13/papers/manysorted.pdf`.

[HB08] Graham Hughes and Tevfik Bultan. Automated verification of access control policies using a SAT solver. *STTT*, 10(6):503–520, 2008.

[Her30] J. Herbrand. *Recherches sur la théorie de la démonstration*. PhD thesis, Université de Paris, Paris, France, 1930.

[HMU06] John E. Hopcroft, R. Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, third edition, 2006.

[HRCS02] JN Hooker, G. Rago, V. Chandru, and A. Shrivastava. Partial instantiation methods for inference in first-order logic. *Journal of Automated Reasoning*, 28(4):371–396, 2002.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, April 2006.

[Jer88] R. G. Jereslow. Computation-oriented reductions of predicate to propositional logic. *Decision Support Systems*, 4:183–197, 1988.

[Lew80] HR Lewis. Complexity results for classes of quantificational formulas. *J. COMP. AND SYS. SCI.*, 21(3):317–353, 1980.

[LG04] Alex X. Liu and Mohamed G. Gouda. Diverse firewall design. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN)*, pages 595–604, 2004.

[LGar] Alex X. Liu and Mohamed G. Gouda. Firewall policy queries. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, to appear.

[Liu07] Alex X. Liu. Change-impact analysis of firewall policies. In *Proceedings of the 12th European Symposium Research Computer Security (ESORICS)*, pages 155–170, 2007.

[LS04] S.K. Lahiri and S.A. Seshia. The UCLID decision procedure. In *16th International Conference, Computer-Aided Verification*, pages 475–478. Springer, 2004.

[Mil02] Scott G. Miller. SISC: A complete scheme interpreter in java. Technical report, 2002.

[MK05a] Robert Marmorstein and Phil Kearns. An open source solution for testing nat'd and nested iptables firewalls. In *LISA '05: Proceedings of the 19th conference on Large Installation System Administration Conference*, pages 11–11, Berkeley, CA, USA, 2005. USENIX Association.

[MK05b] Robert Marmorstein and Phil Kearns. A tool for automated iptables firewall analysis. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 44–44, Berkeley, CA, USA, 2005. USENIX Association.

[MK06] Robert Marmorstein and Phil Kearns. Firewall analysis with policy-based host classification. In *LISA '06: Proceedings of the 20th conference on Large Installation System Administration*, Berkeley, CA, USA, 2006. USENIX Association.

[Mom05] Lee Momtahan. Towards a small model theorem for data independent systems in Alloy. *Electronic Notes in Theoretical Computer Science*, 128(6):37 – 52, 2005. Proceedings of the Fouth International Workshop on Automated Verification of Critical Systems (AVoCS 2004).

[MWZ00] Alain Mayer, Avishai Wool, and Elisha Ziskind. Fang: A firewall analysis engine. In *Proc. 2000 IEEE Symposium on Security and Privacy*, pages 177–187, 2000.

[MWZ05] Alain Mayer, Avishai Wool, and Elisha Ziskind. Offline firewall analysis. *Int. J. Inf. Secur.*, 2005.

[OAS05] OASIS. Oasis extensible access control markup language, 2005. http://www.oasis-open.org/committees/xacml.

[Obe89] A. Oberschelp. Order sorted predicate logic. In *Workshop on Sorts and Types in Artificial Intelligence*, pages 1–17. Springer, 1989.

[OGP03] David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do internet services fail, and what can be done about it? In *4th Usenix Symposium on Internet Technologies and Systems*, 2003.

[OLK09] Ricardo M. Oliveira, Sihyung Lee, and Hyong S. Kim. Automatic detection of firewall misconfigurations using firewall and network routing policies. In *DSN 2009 Workshop on Proactive Failure Avoidance, Recovery and Maintenance*, 2009.

[PdMB08] Ruzica Piskac, Leonardo de Moura, and Nikolaj Bjørner. Deciding Effectively Propositional Logic with Equality. Technical Report MSR-TR-2008-181, Microsoft Research, December 2008.

[Ram30] Frank P. Ramsey. On a problem in formal logic. *Proceedings of the London Mathematical Society*, 30:264–286, 1930.

[SNGM89] Gert Smolka, Werner Nutt, Joseph A. Goguen, and José Meseguer. Order-sorted equational computation. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 2: Rewriting Techniques, pages 299–369. Academic Press, New York, 1989.

[TJ07] E. Torlak and D. Jackson. Kodkod: A relational model finder. 2007.

[VP05] Pavan Verma and Atul Prakash. FACE: A Firewall Analysis and Configuration Engine. In *Proceedings of the 2005 Symposium on Applications and the Internet (SAINT '05)*, 2005.

[Wal88] C. Walther. Many-sorted unification. *Journal of the ACM (JACM)*, 35(1):1–17, 1988.

[Wei97] T. Weibel. An order-sorted resolution in theory and practice. *Theoretical computer science*, 185(2):393–410, 1997.

[Woo01] Avishai Wool. Architecting the lumeta firewall analyzer. In *Proceedings of the 10th USENIX Security Symposium*, 2001.

[Woo04] Avishai Wool. A quantitative study of firewall configuration errors. *Computer*, 37, no. 6:62–67, 2004.

[YMS+06] L. Yuan, J. Mai, Z. Su, H. Chen, C-N. Chuah, and P. Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *IEEE Symposium on Security and Privacy*, 2006.