18734 Homework 1
Privacy Audits with REDUCE

Released: September 11, 2017
Due: 12 noon on September 20 Pacific Time, 2017

Notes:

- **Please start early.** In previous versions of this class, people have typically had the most trouble with this homework.
- Read *every component* of this homework carefully, **including the writeup** before the problems. The logic that REDUCE is different from conventional first-order logic. The differents are outlined in the first part.
- The actual homework problems begin on page 10.

1 Background on REDUCE [2]

REDUCE is an algorithm that checks audit logs for compliance with privacy policies. For instance, REDUCE could be used to ensure that health-care providers comply with HIPAA, the privacy law that limits access to individuals' health information.

1.1 Ensuring that REDUCE terminates

In order for the REDUCE algorithm to terminate in a reasonable amount of time, REDUCE operates on a subset of first-order logic that allows restricted quantification over infinite domains [2].

Thus, REDUCE is only able to check policies that restrict quantifiers. As seen in class, many HIPAA clauses have the form

$$\forall p_1, p_2, m.\ (\mathsf{send}(p_1, p_2, m) \implies \varphi)$$

where p_1 and p_2 are people/organizations, and m is a message that p_1 sends to p_2.

The formula quantifiers over the infinite set of messages. If REDUCE were to naively check for all possible values m can take on, it would not terminate. Fortunately, the number of messages

a hospital sends is finite. Thus, the predicate $\mathsf{send}(p_1, p_2, m)$ is true only for a finite number of possible messages m. Similarly, there are a finite number of candidates for p_1 and p_2.

To ensure every quantified variable is finite so that REDUCE terminates, the algorithm uses mode checking [1].

1.2 Checking for compliance in the face of incomplete information

Logs in the real world often do not contain enough information to decide whether or not the policy has been violated. This is partly because privacy laws often allow disclosures based on subjective beliefs.

For example, a hospital is allowed to share health information with law enforcement officers if the hospital believes that a patient they are treating may have committed a criminal act. Thus, a completely automated enforcement mechanism is not possible. Instead, REDUCE checks as much of the policy as possible over the available information in the log, and outputs a residual policy that it cannot verify.

In order to check this residual policy, the log may need to be extended with additional information, or human analysis may be needed.

2 Specifying policies in REDUCE

2.1 Using guards to limit the number of possible values x can take on in the quantified formulas "$\forall x\ .\varphi$" and $\exists x.\ \varphi$

REDUCE checks policies in a first-order logic with restricted quantifiers, also known as guarded quantifiers.

Because the first-order quantifiers "$\forall$" and "$\exists$" may range over infinitely large domains, these quantifies are restricted to the forms

$$\forall \vec{x}.\ (c(\vec{x}) \supset b(\vec{x}))$$

and

$$\exists \vec{x}.\ (c(\vec{x}) \wedge b(\vec{x}))$$

The formula $c(\vec{x})$ is called a *guard*.

To ensure the REDUCE algorithm terminates, only a finite number of substituions for the quantifier variables $\vec{x}$ are allowed. To enforce this, *mode analysis* is used to ensure the guard $c(\vec{x})$ can only take on a finite number of possible values.

2.2 Example policy specification

In the lecture on September 11, we formalized the following HIPAA policy into a format that REDUCE can check:

> *A covered entity may disclose an individual's protected health information to law-enforcement officials for the purpose of identifying an individual if the individual made a statement admitting participation in a violent crime that the covered entity believes may have caused serious physical harm to the victim.*

To formalize this policy the restricted first-order logic that REDUCE users, we start by assuming that all transmissions made by a covered entity are recorded in a log.

The predicate $\texttt{send}(p_1, p_2, m)$ is true if the transmission of message m from principal p_1 to principal p_2 occurs in this log.

Similarly, we assume that statements made by individuals are also recorded in the log. The predicate $\texttt{state}(q, s)$ means that individual q made a statement s. If statement s made by individual q appears in the log, then $\texttt{state}(q, s)$ is true.

We also assume that each transmitted message m is tagged by the sender with the names of individuals whose information it carries, as well as the attributes of information it carries. These attributes could include be "*address*", "*social-security-number*", "*medications*", "*medical-history*", etc. Tags may or may not be accurate.

Message m is also assumed to be tagged with a purpose u, which is the point of sending the message. This purpose could be "*alert-law-enforcement*", "*id-criminal*", "*notify-allergies*", etc. This is represented by the predicate $\texttt{purp\_in}(u, \mu)$

Attributes and purposes are assumed to have separate hierarchies. For example, the attribute "*medications*" is contained in "*medical-history*". This is formalized as the predicate $\texttt{attr\_in}$ (*medications*, *medical-history*). Similarly, the predicate $\texttt{purp\_in}(u, \mu)$ means that purpose μ is a special case of purpose u. For instance, a surgery is a particular type of medical treatment, so we write $\texttt{purp\_in}$(*treatment*, *surgery*)

We write $\texttt{inrole}(p, r)$ to mean that principal p has role r. For example, $\texttt{inrole}(p,$ *covered-entity*$)$ means that principal p is a covered entity who is required to obey the HIPAA laws.

The predicate $\texttt{is-admission-of-crime}(s)$ means that statement s is an admission of a crime, and predicate $\texttt{believes-crime-caused-serious-harm}(p_1, q, s)$ means that principal p_1 believes that q made a statement s that suggests that q committed a crime that caused harm.

The vocabulary used to formalize this clause is summarized in Table 1 and Table 2.2.

Type	**Constant**	**English**
Information	ϕ	protected health information
Role	*law-enforcement-official*	law enforcement official
Purpose	*id-criminal*	identify a criminal

Table 1: Constants used to formalize a HIPAA clause about permitting disclosure of health information to identify a dangerous criminal

Predicate	**English**
$\texttt{send}(p_1, p_2, m)$	p_1 sends message m to p_2
$\texttt{tagged}(m, q, t, u)$	m is a message containing information with attributes t about q with purpose u
$\texttt{inrole}(p_2, \textit{law-enforcement-official})$	p_2 has the role of a law-enforcement official
$\texttt{attr\_in}(t, \mathcal{I})$	t contains information $\mathcal{I}$
$\texttt{purp\_in}(u, \textit{id-criminal})$	purpose u is to identify a criminal
$\texttt{state}(q, s)$	q states s
$\texttt{is-admission-of-crime}(s)$	s is an admission of crime
$\texttt{believes-crime-caused-serious-harm}(p, q, s)$	p believes q may have caused serious harm

Table 2: Predicates used to formalize a HIPAA clause about permitting disclosure of health information to identify a dangerous criminal

The clause is formalized as follows:

$$
\begin{aligned}
&\forall p_1, p_2, m, q, t, \\
&\quad \texttt{send}(p_1, p_2, m) \\
&\quad \wedge \texttt{tagged}(m, q, t, u) \\
&\quad \wedge \texttt{attr\_in}(t, \phi) \\
&\quad \supset \\
&\qquad \texttt{inrole}(p_1, \textit{covered-entity}) \\
&\qquad \wedge \texttt{inrole}(p_2, \textit{law-enforcement-official}) \\
&\qquad \wedge \texttt{purp-in}(u, \textit{id-criminal}) \\
&\qquad \wedge \exists s \text{ s.t.} \\
&\qquad\quad \texttt{state}(q, s) \\
&\qquad\quad \wedge \texttt{is-admission-of-crime}(s) \\
&\qquad\quad \wedge \texttt{believes-crime-caused-serious-harm}(p_1, q, s)
\end{aligned}
$$

In words, entity p_1 sends to entity p_2 a message m, and message m is tagged as carrying attribute t of individual q's protected health information ϕ, and is also tagged with purpose u, then p_1 is a covered entity who is required to obey HIPAA, p_2 is a law enforcement official, message m was sent for the purpose of identifying a criminal, and individual q made statement that is an admission of a crime, and furthermore entity p_1 believes that this statement implies that the crime q claimed he made, cause harm.

REDUCE verifies this clause as follows:

- Predicates $\texttt{send}(p_1, p_2, m)$ and $\texttt{state}(q, s)$ are verified by looking up in their respective logs.
- Predicates $\texttt{tagged}(m, q, t, u)$ and $\texttt{purp\_in}(u, \textit{id-criminal})$ are verified by examining the tags in message m through a pre-defined computable function.
- Predicates $\texttt{attr\_in}(t, \mathcal{T})$ and $\texttt{purp\_in}(u, \mu)$ are verified by a function that checks stipulated hierarchies over attributes and purposes respectively.

- The predicates `is-admission-of-crime`(s) and `believes-crime-caused-serious-harm`(p_1, q, s) may require human input to resolve, as it whether or not it is reasonable to believe that statement s suggests a crime was committed and caused serious harm, is something that may vary depending on which person you ask.

The REDUCE tool requires the policy designer to categorize predicates based on how they are verified.

2.3 Mode analysis

A naive way to check a policy of the form $\forall x.\ b(x)$ is to instantiate x with every possible element of domain D that x can range over, then checks if $b(x)$ is true or false based on that instance of x. If domain D were infinite, this leads to non-termination.

Real policies may quantify over infinite domains. For example, HIPAA contains quantification over the infinite domain of messages. Thus, something must be done to limit the number of relevant instances of x that must be checked.

As mentioned in the lecture on September 11, REDUCE relies on the restrictions c (also known as the *guard*) in the quantified formulas $\forall x.\ c(x) \supset b(x)$ and $\exists x.\ c(x) \wedge b(x)$. It uses *mode analysis* to ensure that the guard c accepts only a finite number of satisfyingargumentss in any formula.

Mode analysis requires the policy designer to specify which argument positions of a predicate can be *derived* from others. In the example in Section 2.2, we assume that given a message m about an individual q, the attributes of the message and the purpose u of sending the message, are already tagged in m.

Thus, we denote the predicate `tagged`(m, q, t, u) to have the mode `tagged`$(+, +, -, -)$. The mode `tagged`$(-, -, -, +)$ is incorrect, because given a fixed argument u (purpose), there may be an infinite number of first arguments m (messages) annotated with that purpose. Thus, the latter set cannot be finitely computed.

Similarly, if the predicate `mult`(x, y, z) means that $x = yz$, where x, y, and z are integers larger than 0, then any of the modes `mult`$(+, +, -)$, `mult`$(-, +, +)$, and `mult`$(+, -, +)$ are ok. This is because given any two of the three variables x, y, and z, we can determined the value of the last variable of $x = yz$ by some simple arithmetic. However, `mult`$(-, -, +)$ is incorrect, as given only one of x, y, and z, there are a infinite number of substitutions for the remaining two variables that make the equation $x = yz$ true.

Given the mode information of all predicates in a policy, a static, linear-time check of the policy, called a *mode check*, ensures that there are only a finite number of instances of free variables that can satisfy a guard c in the policy.

2.4 Designing the guard

To understand which predicates can be used in the guard, it is important to understand variable modes in predicates. You can find variable modes in addition to other information in the file

`hipaa.d`. Some rules for designing the guard and information about where to find mode information follows:

1. The guard cannot contain a subjective predicate. If a predicate is subjective, you can see it in the `hipaa.d` file. For example,

   ```
   is-valid-authorization:
     message(+) -> prin(+) -> prin(+) -> prin(+) -> date(+) -> pred SUBJ.
   ```

   is subjective predicate as indicated by `pred SUBJ` at the end of the definition.

2. The guard can contain a predicate which has all variables in the output mode. You can find the mode of the variables in a predicate beside the variables in the predicate definition in the hipaa.d file. For example,

   ```
   send:  prin(-) -> prin(-) -> message(-) -> date(-) -> pred DB.
   ```

   can be used in a guard because all the variables `(prin, prin, message, date)` are in output mode. `pred DB` indicates that REDUCE finds finite substitutions by searching through a database.

3. If a predicate has a variable in input mode, it must be preceded by another predicate which has the same variable in output mode. For example, if look at the `send` and `eq_ msg` predicates

   ```
   send:  prin(-) -> prin(-) -> message(-) -> date(-) -> pred DB
   ```

   and

   ```
   eq_ msg:  message(+) -> message(-) -> pred EVAL.
   ```

   then

   $all[p1][p2][m][i][u][pp]$
   $(and$
   $(send\ p1\ p2\ m\ u)$
   $(eq\_msg\ m\ (msg\ i\ pp))$
)
   (true)

   is a valid policy statement, because m is in input mode in eq_msg, but in output mode in $send$.

An interesting example is

```
inrelation:  prin(+) -> prin(+) -> relation(-) -> date(-) -> pred DB.
```

In the provided `hipaa.f` file, you would find that `inrelation` is used in the guard despite having two variables in the input mode:

$ex[u1]$
$(inrelation\ p\ p2\ treatment{-}relation\ u1)$
$(true)$

This is because the guard of $ex[u1]$ requires finite substitutions of the variable being quantified, which is u_1. However, the variables p and p_2 that are in input mode are not quantified here.

An important point to remember (particularly for Problem 2) is that

$$\exists x, y.\ c(x, y) \wedge b(x, y) \wedge a(x, y)$$

must be written as

$ex[x][y]$
  $(c(x, y))$
  $(and\ b(x, y)\ a(x, y))$

as ex only takes two predicates as arguments. Further remember that c is a guard and hence c must have finite substitutions. When checking the audit log the reduce algorithm checks the guard first.

2.5 Negation in REDUCE

For technical reasons, negation is not included in the logic REDUCE uses. Interested students can refer to Garg et al 2011 [2] for details on how to reason about negation in REDUCE.

This homework does not require the use of negation.

2.6 Notation

In REDUCE, prefix notation is used. For instance,

$$a \wedge b$$

is written as

(and (a) (b))

Following the restrictions on the quantifiers "$\forall$" and "$\exists$" as described above,

$$\forall x.\ c(x) \supset b(x)$$

is written as

$all[x]$
  $(c(x))$
  $(b(x))$

with the implication "$\supset$" being implicit (Note the brackets carefully).

Table 2.6 specifies the syntax mapping from first-order logic to the language that REDUCE uses.

First-Order Logic	REDUCE Language
$a \wedge b$	$and\ (a)\ (b)$
$a \vee b$	$or\ (a)\ (b)$
$a \supset b$	$imp\ (a)\ (b)$
$a + b$	$plus\ (a)\ (b)$
$\forall x, y.\ c(x, y) \supset b(x, y)$	$all\ [x][y]\ (c(x, y))\ (b(x, y))$
$\exists x, y.\ c(x, y) \wedge b(x, y)$	$ex\ [x][y]\ (c(x, y))\ (b(x, y))$
predicate-name$(arg1, \ldots, argN)$	$(predicate{-}name\ arg1\ \cdots\ argN)$

Table 3: Syntax mapping from first-order logic to language used by REDUCE

3 Workflow for REDUCE

3.1 Software

You can either do the homework on your own Linux machine or on our sever.

In order to do it on your machine, install sqlite3 and download all the necessary files from `https://www.ece.cmu.edu/~ece734/homeworks/hw1-files.zip`.

In order to do it on our server you need to:

1. Download and install Cisco AnyConnect for connecting to the campus VPN: `https://www.cmu.edu/computing/services/endpoint/network-access/vpn/`.
2. Connect to vpn.sv.cmu.edu with your Andrew ID and password.
3. SSH to yourandrewID@10.0.21.116. You password will be the same as your login (i.e. your Andrew ID). You can change it by typing passwd in the terminal.

If you choose to work on the server, all the necessary software will be already installed, and all the necessary files will be in your home directory.

You can use scp/sftp to put or get files from the server if you are not comfortable editing with linux editors. If you are using Windows, The Tectia client [1] allows you to ssh into the server, and also copy files to and from the server to your computer.

Be careful about the end of line characters that differ on Windows and Linux. Notepad++ is a good Windows editor that allows converting Windows EOL to Linux EOL.

The binary file for the tool named "`test-hipaa`" has been provided in the home directory. The file containing the policy is stored in `$HOME/hipaa-case-study/hipaa.f` and the file containing various configurations necessary for `test-hipaa` is stored in `$HOME/hipaa-case-study/hipaa.d`. The database file that stores the log must be in the `$HOME` directory and must be named `hipaa.db`.

[1] `http://www.cmu.edu/computing/software/all/tectia/index.html`

3.2 Steps to run REDUCE

Files needed to run REDUCE:

The folder `hipaa-case-study` contains two files: `hipaa.f` and `hippa.d`.

`hipaa.f` is the policy file that specifies a policy in REDUCE. For each problem, you have to convert a given clause(s) into the language for REDUCE, and add it to `hipaa.f`.

You are *not* expected to modify `hippa.d`.

Testing your policy specification:

To test your policy specification, you would require audit logs, which are essentially database (`hipaa.db`) files. In your parent directory, you will find several `.sql` files which can generate appropriate audit logs. You can generate audit logs by running

`sqlite3 hipaa.db` < ⟨`filename`⟩`.sql`

Remember that the database file MUST be named `hipaa.db`, and must be located in the parent directory for the tool to work.

Also, in order to test every new log, you MUST remove the old `hipaa.db` file. Otherwise, the sql script will append to the existing file, and you won't get expected results.

Running to tool:

To run the REDUCE tool, type

`./test-hipaa`

from the command prompt.

The tool then uses the policy file (`hipaa.f`), the database file (`hipaa.db`), and the specifications file (`hipaa.d`) to produce an output.

Expected forms of output:

When the tool is run output is either audit passed (true) or audit failed (false). The first line of output is the time taken, followed by (true or false) or a subjective predicate in some cases.

For example, if the audit log shows that the policy was not obeyed, the output would look similar to this:

```
%%%%%%%%% time used:  0.001 %%%%%%%%%%%%
⟨POL/DISCLOSE⟩ false)
```

If REDUCE is not able to determine whether the policy was obeyed due to incomplete information in the log / the use of subjective predicates, the output would look similar to this:

```
%%%%%%%%% time used:  0.001 %%%%%%%%%%%%
(is-valid-authorization (str2msg ‘‘_fake’’) (str2prin ‘‘_hospital0’’)
(str2prin ‘‘_rv1’’) (str2prin ‘‘_p1’’) 1971:04:01:20:30:00)
```

Problem 1 [10]

a) Understanding notation

1. Translate the given hipaa.f file into a logic formula and write it out. The predicates are send, hasattrof, eq_ msg, inrelation, time_in, refers. This is a policy with two positive norms.
2. Next, with the following interpretations, explain the policy in English.
   (send p1 p2 m u) means that "p1" is sender of message "m" to "p2" at time "u".
   (eq_msg m (msg i pp)) means that message "m" is the pair (i, pp), where "i" is info, "pp" is purpose.
   (hasattrof i p t) means info "i" contains attributes "t" about individual "p".
   (inrelation p p2 treatment-relation u1) means "p" has treatment-relation with "p2" at time "u1".
   (time_in u v w) means $u < v < w$, where time is counted in days.
   (refers p1 p2 p referral u1) mean "p1" referred individual "p" to "p2" at time "u1".
   (plus u 365) is a function that returns u+365. $\sim$ indicates negative (-). Thus (plus u $\sim$ 365) returns $u - 365$.

Produce a log (hipaa.db) by executing "sqlite3 hipaa.db <prob1.sql". Upon running the tool by executing "./test-hipaa", the audit should fail, i.e. output false.

b) Adding a positive norm

Using the predicate:
(prescribes p2 q prescription w) meaning "p2" prescribes a prescription for patient "q" at time "w"
add a positive norm that says that "p2" prescribes a prescription for patient "q" within the next 30 days of the day "u" when the message "m" was sent.

To test if your written policy is correct run the tool again with the modified hipaa.f that contains the added positive norm. The log contains a message that is allowed by the added positive norm. So the audit succeeds, i.e. returns true.

Problem 2 [20]

Using the following predicates:

(attr_in t psychotherapy-notes) meaning attributes "t" contains the attribute "psychotherapy-notes"

(lessthan u t) meaning $u < t$

(is-valid-authorization m p1 p2 p u) meaning "m" is a valid authorization by "p" for "p1" to enable him sending message with "psychotherapy-notes" about "p" to 'p2" at time "u"

add a negative norm stating that in cases when the message "m" sent by a covered-entity "p1" to any entity "p2" contained "psychotherapy-notes" about "p", then it must be the case that "p1" had, in the past, received a message from "p" that was a valid authorization by "p".

There are two sql files provided for this problem: prob2fake.sql and prob2.sql. The log generated by prob2fake.sql contains a fake authorization sent by "p" to "p1" and the log in prob2.sql contains no sending of any such authorization. Further, it is specified in hipaa.d that the predicate is-valid-authorization is a subjective predicate.

To test if your written policy is correct, run the tool on the log generated by prob2fake.sql and prob2.sql (one at a time) with your modified hipaa.f that contains the negative norm. Then, if the negative norm has been written properly, with prob2.sql the audit will just fail and with prob2fake.sql the audit will output the subjective predicate is-valid-authorization. You should delete the existing hipaa.db file (using the command "rm hipaa.db") before creating new ones, otherwise entries keep getting appended to the existing db file and you may get unexpected results.

Problem 3 [20]

For the given clause, do the following:

1. Identify the type of clause (positive or negative)
2. Write the clause in logic (predicates are provided below)
3. Put the clause in the hipaa.f file. An sql file is provided to test the correctness of your implementation.

A covered entity that participates in an organized health care arrangement may disclose protected health information about an individual to another covered entity that participates in the organized health care arrangement for any health care operations activities of the organized health care arrangement.

The predicates (functions) to you'll need are described below:

(inrole p role u) meaning p is in role "role" at time u.

(belongrole p role u) p is in role "role" at time u

(The above is a subjective predicate as opposed to inrole, which is known from a db)
"covered-entity", "organized-healthcare-arrangement" are roles that you will need to use. "covered-entity" is a recognized role, whereas "organized-healthcare-arrangement" is not a recognized role.

(purp_in pp purpose) meaning pp contains the purpose "purpose".

(is-participant-of-organized-health p q u) p is a participant in the organized health-care arrangement provided by q at time u

(healthcare-op p) is a function returning a purpose—the returned purpose is the health care operations of p.

"phi" is a known attribute meaning "protected health information".

The log created by prob3.sql should output the subjective predicate for the first clause (belongrole). That means that this log passed all checks of the first clause, except for the subjective predicate that the tool does not know how to evaluate.

Problem 4 [30]

For the given clause, do the following:

1. Identify the type of clause (positive or negative)
2. Write the clause in logic (predicates are provided below)
3. Put the clause in the hipaa.f file. An sql file is provided to test the correctness of your implementation.

A covered entity may disclose protected health information to a public health authority that is authorized by law to collect or receive such information for the purpose of preventing or controlling disease; or, at the direction of such an authorized public health authority, to a foreign government agency.

Two concrete roles to be used are "public-health-authority" and "foreign-gov-agency". One concrete purpose is "disease-prevention-control".

(is-authorized-by-law p1 pp u) meaning p1 is authorized by the law to collect or receive protected health information. (This is a subjective predicate)

(directed-disclosure p0 p1 p2 t pp u) meaning p0 has directed p1 to disclose to p2 information that contains the attribute t with purpose pp at time u.

The log created by prob4.sql should output one subjective predicate (is-authorized-by-law). That means that the log created by prob4.sql passed all checks, except for the subjective predicate that the tool does not know how to evaluate.

Problem 5 [20]

In this problem we extend the policy by adding exceptions to clauses (as described in class). The deliverables of this part is

1. Identify the type of clause (positive or negative)
2. Write the clause in logic (predicates are provided below)
3. Put the clause in the hipaa.f file. An sql file is provided to test the correctness of your implementation.

Notwithstanding any provision of this subpart, a covered entity must obtain an authorization for any use or disclosure of psychotherapy notes, except to carry out the following treatment, payment, or health care operations:
(A) Use by the originator of the psychotherapy notes for treatment;
(B) Use or disclosure by the covered entity for its own training programs in which students, trainees, or practitioners in mental health learn under supervision to practice or improve their skills in group, joint, family, or individual counseling;
or
(C) Use or disclosure by the covered entity to defend itself in a legal action or other proceeding brought by the individual

The predicates (functions) to be used in each clause are described below

(is-valid-authorization m p1 p2 p u) "m" s a valid authorization by "p" for "p1" to enable him sending message with "psychotherapy-notes" about "p" to 'p2" at time "u"
(counseling-training-programs p) is a function returning a purpose—the returned purpose is counseling training programs of p.
(defense-in-legal-proceeding p q) is a function returning a purpose—the returned purpose is for defense of p in a legal case initiated by q.

One concrete attribute is "psychotherapy-notes".

User the provided prob5.sql to create a log file. This log contains an authorization by patient followed by sending of message with psychotherapy notes by the hospital. The sends are so formulated such that on running the tool, the output will be two "is-valid-authorization" predicates.

Next, uncomment line 55 (remove leading $--$) in prob5.sql and generate the log again. This line adds the purpose counseling and training. Now, the audit will succeed, and the tool will output 'true'.

Files to be submitted

Make a pdf file for the written parts. Put the hipaa.f and the pdf files in one zip file, name the file <your_andrew_id>_HW1.zip and submit on canvas. The submission deadline is 12 noon (not midnight), Sep 20.

References

[1] K. R. Apt and E. Marchiori. Reasoning about prolog programs: from modes through types to assertions. *Formal aspects of computing*, 6(1):743–765, 1994.

[2] D. Garg, L. Jia, and A. Datta. Policy auditing over incomplete logs: theory, implementation and applications. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 151–162. ACM, 2011.