Rewrite Verification System

Siddhartha Prasad

30th July 2014

1 Summary

This report presents the design of a certified rewrite and equality verification system. It also details the structure of 'proof certificates' that equality checkers and rewriters must produce to allow for the correctness of a proof to be verified independently of the proving system.

Code and documentation can be found at `https://github.com/sidprasad/RewriteVerificationSystem`

Contents

2 Introduction

Given a set Δ of implicitly universally quantified equations in a first-order equational logic, and two terms x and y, the provability of the equation $(x = y)$ from Δ is defined by the following rules [1] [3]:

$$\text{Axiom}\ \frac{(x = y) \in \Delta}{\Delta \vdash (x = y)}$$

$$\text{Instantiation}\ \frac{\Delta \vdash (x = y)}{\Delta \vdash \mathit{subst}\ i(x = y)}$$

$$\text{Refl.}\ \frac{}{\Delta \vdash (x = x)}$$

$$\text{Sym.}\ \frac{\Delta \vdash (x = y)}{\Delta \vdash (y = x)}$$

$$\text{Trans.}\ \frac{\Delta \vdash (x = x') \qquad \Delta \vdash (x' = y)}{\Delta \vdash (x = y)}$$

$$\text{Cong.}\ \frac{\Delta \vdash (x_1 = y_1) \qquad \ldots \qquad \Delta \vdash (x_n = y_n)}{\Delta \vdash (f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n))}$$

Thus, $x = y$ holds in all normal models of Δ iff the proof of $x = y$ is a 'tree' that is definable by repeated use of these rules. Such proof trees can be large, and it is unlikely that implementors of rewrite systems will provide 'certificates' of correctness with respect to the equality theory in such detail.

The aim of this report is to present a verifier which we claim can, given a smaller 'certificate', reconstruct such a proof of equality. This will allow for the use of certifiable, but not necessarily certified, rewrite and equality checking systems.

3 Proof Certificate Structure

3.1 Components of a Certificate

The proof certificates accepted by the verifier are flexible in their syntax. For an equality $s = t$, where the proof involves rewriting s to t, the certificate must include:

1. The terms s and t.

2. A certificate list that contains the justifications that an equality proof is valid. This could be of the form:

   (a) A list of the rewrite rules used, in order.

(b) A list of the subterms of s (or any intermediate superterm) to be rewritten, in the order they were carried out.

(c) Some combination of (a) and (b).

(d) The total number of rewrite rules used.

3. The set of rewrite rules and tree-building rules P_Δ and P_Σ as described later.

3.2 Tacticals

The system also contains a set of 'tacticals' that allow for manipulation of rewrite rules.

1. The "else" tactical accepts two rewrite rules and a term. It attempts to apply the first rewrite rule to the term. If it fails, it attempts to apply the second rule to the term.

2. The "sym" tactical applies the symmetric version of a rewrite rule to a term.

3. The "conv" tactical applies the converse of a rewrite rule to a term.

4. The "then" tactical takes two terms with a common 'ancestor' in the 'super-term', and applies a rewrite rule to each without affecting anything higher in the term than the common ancestor.

These tacticals allow for greater flexibility for the user, and also reduce the size of certificates required by the verifier.

The implementation details and syntax of the proof certificates can be found in the rewrite verification system's 'README' file. An example certificate can also be found in Appendix A.3.

4 Implementation

The verifier is implemented in SWI-Prolog.

1. As the language is logical and declarative in nature, the correctness proof of the verifier is very similar to the system's source code.

2. The structure of an internal proof constructed by the system is also evident from source code.

3. The modular nature of the language allows for proof certificate rules to be easily used by the verifier.

More specific implementation notes can be found in the README file.

5 Correctness of the Verifier

Let Δ be the set of (implicitly universally quantified) 'rewrite rules' in the rewrite system, and P be the verifier program.

$$\Delta = \{(s_1 = t_1), (s_2 = t_2), \ldots, (s_n = t_n)\}$$

Let P_Δ be a subset of P, defined as follows:
$P_\Delta = \{\mathit{Pred}\ s_1\ t_1,\ \mathit{Pred}\ s_2\ t_2, \ldots, \mathit{Pred}\ s_n\ t_n\} \cup$ the set of inbuilt Prolog predicates.

Here the predicate *Pred* implements the rewriting steps in an equality proof.

Atomic formulas using the predicates $\{\mathit{isList}, \mathit{notList}, \mathit{member}, \mathit{listmem}, \mathit{replace}\}$ can all be proved from the set of inbuilt Prolog predicates, and so from P_Δ. The proof of this can be found in Appendix A.1. As they hold, and do not appreciably contribute to the Prolog proof, they are not explicitly mentioned in the following proofs.

Let P_Δ be related to Δ as follows:
For $\mathit{Pred} \in P_\Delta$, $\vdash \mathit{Pred}\ s\ t \Rightarrow \Delta \vdash (s = t)$. Thus, a predicate *Pred* can be considered Δ-sound iff $P_\Delta \vdash \mathit{Pred}\ t\ s \ \Rightarrow\ \Delta \vdash (t = s)$.
The corresponding proof tree of $\Delta \vdash (t = s)$ is of the form:

$$
\begin{array}{rc}
\text{Axiom} & \dfrac{(\hat{s} = \hat{t}) \in \Delta}{\Delta \vdash \hat{s} = \hat{t}} \\
\cline{2-2}
k \text{ Instantiations} & \dfrac{\vdots}{\vdots} \\
\cline{2-2}
 & \vdots \\
\text{Instantiation} & \dfrac{}{\Delta \vdash \mathit{subst}\ i(s = t)}
\end{array}
$$

Certificates of correctness are expressed using trees rather than terms. Isomorphic to the above relationship, we have the 'tree'-structured counterparts. Let Σ be the set of 'treeify' equalities in the certificate. Let P_Σ be a subset of P related to Σ as follows: If $\vdash P_\Sigma\ t_2\ t_1$ then $\Sigma \vdash (t_1 = t_2)$.
The corresponding proof tree is of the form:

$$
\begin{array}{rc}
\text{Axiom} & \dfrac{(\hat{t_1} = \hat{t_2}) \in \Sigma}{\Sigma \vdash \hat{t_1} = \hat{t_2}} \\
\cline{2-2}
k' \text{ Instantiations} & \dfrac{\vdots}{\vdots} \\
\cline{2-2}
 & \vdots \\
\text{Instantiation} & \dfrac{}{\Sigma \vdash \mathit{subst}\ i(t_1 = t_2)}
\end{array}
$$

P_Σ is represented by the 'treeify' rules in the example certificate in Appendix A.3 .

Let the set of all predicates and rules related to the program P be denoted by Γ.

$$P = P_\Delta \cup P_\Sigma \cup \{\mathit{isList}, \mathit{notList}, \mathit{member}, \mathit{listmem}, \mathit{replace}\} \cup \{p \mid p \text{ is an inbuilt Prolog predicate}\}$$

In such a system, the following theorems hold for the program P:

```
%Allows for a converse relation
conv(Pred, T, S) :- G=..[Pred, S, T], G.
```

Lemma 1. *Let Pred be Δ-sound, and G be "conv Pred t s". Then,* $P_\Delta \vdash G \;\Rightarrow\; \Delta \vdash (t = s)$.

Proof. By inspection of the Prolog proof:
If $P_\Delta \vdash G$ then $P_\Delta \vdash \mathit{Pred}\ s\ t$.

By the definition of the relationship between P_Δ and Δ, if $P_\Delta \vdash \mathit{Pred}\ s\ t$ then $\Delta \vdash (s = t)$.

$$\text{SYM}\ \frac{\Delta \vdash (s = t)}{\Delta \vdash (t = s)}$$

Thus, $P_\Delta \vdash G \Rightarrow \Delta \vdash (t = s)$.

□

```
%Allows for a predicate to be made symmetric
sym(Pred, T, S) :- G=..[Pred, T, S],G.
sym(Pred, T, S) :-  G=..[Pred, S, T], G.
```

Lemma 2. *Let Pred be Δ-sound, and G be "sym Pred t s". Then,* $P_\Delta \vdash G \Rightarrow \Delta \vdash (t = s)$.

Proof. By inspection of the Prolog proof:
If $P_\Delta \vdash G$ then $P_\Delta \vdash \mathit{Pred}\ t\ s$ or $P_\Delta \vdash \mathit{Pred}\ s\ t$.

Consider the following cases:

1. If $P_\Delta \vdash \mathit{Pred}\ t\ s$ then $\Delta \vdash (t = s)$
2. If $P_\Delta \vdash \mathit{Pred}\ s\ t$ then

$$\text{SYM}\ \frac{\Delta \vdash (s = t)}{\Delta \vdash (t = s)}$$

Thus, $P_\Delta \vdash G \Rightarrow \Delta \vdash (t = s)$.

□

```
%Allows for Pred1(T,S) to be tried, and Pred2(T,S) if it doesn't work
else_((Pred1,Pred2), T, S) :- (G=..[Pred1,  T, S], G );(G=..[Pred2, T, S],G).
```

Lemma 3. *Let $Pred_1$ and $Pred_2$ be Δ-sound, and G be "else $(Pred_1, Pred_2)$ t s". Then, $P_\Delta \vdash G \;\Rightarrow\; \Delta \vdash (t = s)$.*

Proof. By inspection of the Prolog proof:
If $P_\Delta \vdash G$ then $P_\Delta \vdash Pred_1\ t\ s$ or $P_\Delta \vdash Pred_2\ t\ s$.
Consider the following cases:

1. If $P_\Delta \vdash Pred_1\ t\ s$, then by definition $\Delta \vdash (t = s)$
2. If $P_\Delta \vdash Pred_2\ t\ s$, then by definition $\Delta \vdash (t = s)$

Thus, $P_\Delta \vdash G \Rightarrow \Delta \vdash (t = s)$.

□

Theorem 1. *Let P'_Δ be a subset of P_Δ that does not involve the "then" tactical. Let Pred be Δ-sound, and G be "applyPred Pred t s". If $P'_\Delta \vdash G$ then $\Delta \vdash (s = t)$.*

Proof: By induction on the Prolog proof of $P'_\Delta \vdash G$:

1. If $(Pred\ s\ t) \in P'_\Delta$:

$$\cfrac{\cfrac{\cfrac{(\hat{s} = \hat{t}) \in \Delta}{\Delta \vdash \hat{s} = \hat{t}}\,\text{Axiom}}{\vdots}\,k''\text{ Instantiations}}{\Delta \vdash subst\ i(s' = t')}\,\text{Instantiation}$$

Thus $P'_\Delta \vdash G \Rightarrow \Delta \vdash (s = t)$.

2. Let $f(x) = OP(x, a_1, ..., a_n)$ where OP is some operation in the term algebra, and $a_1, .., a_n$ are terms. Thus, x is a sub-term of the term $f(x)$.

   Let G be "*applyPred Pred* t' s'". where t' and s' are some terms and $(Pred\ s'\ t') \notin P'_\Delta$.

   Thus, if $P'_\Delta \vdash G'$ then $P'_\Delta \vdash applyPred\ Pred\ t\ s$ where t is some sub-term of t' and s is some sub-term of s'.

   By the inductive assumption: $\Delta \vdash (s = t)$.

   We can look at s' as $f(s)$ and t' as $f(t)$. Thus:

$$\text{CONG}\ \frac{\Delta \vdash s = t}{\Delta \vdash f(s) = f(t)}$$

   Thus, $P'_\Delta \vdash G \Rightarrow \Delta \vdash (s' = t')$.

Thus, inductively proved that if $P'_\Delta \vdash G$ then $\Delta \vdash (s = t)$.

Lemma 4. *Let $Pred_1$ and $Pred_2$ be Δ-sound, P_1 and P_2 be paths of subterms from a common ancestor and G be "$then((Pred_1, P_1), (Pred_2, P_2))\ t\ s$". Then, $P_\Delta \vdash G \Rightarrow \Delta \vdash (t = s)$.*

Proof. By induction on Prolog proof:

If *Pred* does not include the *then* tactical: If $P_\Delta \vdash G$ then $P_\Delta \vdash applyPred\ Pred\ t\ t'$ and $P_\Delta \vdash applyPred\ Pred\ t'\ s$.

Thus, by Theorem 1, $\Delta \vdash (t = t')$ and $\Delta \vdash s = t'$.

$$\text{TRANS}\ \frac{\Delta \vdash (t = t') \qquad \text{SYM}\ \dfrac{\Delta \vdash (s = t')}{\Delta \vdash (t' = s)}}{\Delta \vdash (t = s)}$$

Thus, $P_\Delta \vdash G \Rightarrow \Delta \vdash (t = s)$.

(Inductive hypothesis): Let $P_\Delta \vdash then((Pred_1, P_1), (Pred_2, P_2))\ t\ s$.
Let G' be *then* (*then*, $((Pred1, P1), (Pred2, P2))\ t\ s$.
Thus, if $P_\Delta \vdash G'$ then $P_\Delta \vdash G$.
Then, by the inductive hypothesis $\Delta \vdash (t = s)$.
Thus, $P_\Delta \vdash G' \Rightarrow \Delta \vdash (t = s)$.

□

Theorem 1 can now be refined to allow for tacticals.

Theorem 2. *Let Pred be Δ-sound, and G be applyPred Pred t s. If $P_\Delta \vdash G$ then $\Delta \vdash (s = t)$.*

Proof. By induction on the Prolog proof of $P_\Delta \vdash G$.

1. (a) If $Pred\ s\ t \in P_\Delta$:

$$\text{Instantiation}\ \frac{n \text{ Instantiations}\ \dfrac{\text{Axiom}\ \dfrac{(\hat{s} = \hat{t}) \in \Delta}{\Delta \vdash \hat{s} = \hat{t}}}{\vdots}}{\Delta \vdash \text{subs } i(s' = t')}$$

   (b) If $P_\Delta \vdash (conv, Pred)\ s\ t$ then $\Delta \vdash (s = t)$. (by Theorem i)

   (c) If $P_\Delta \vdash (sym, Pred)\ s\ t$ then $\Delta \vdash (s = t)$. (by Theorem ii)

   (d) If $P_\Delta \vdash (else, (Pred1, Pred2))\ s\ t$ then $\Delta \vdash (s = t)$. (by Theorem iii)

(e) If $P_\Delta \vdash (then, ((Pred1, P1), (Pred2, P2))\ s\ t$ then $\Delta \vdash (s = t)$. (by Theorem iv)

Thus $P_\Delta \vdash G \Rightarrow \Delta \vdash (s = t)$.

2. Let $f(x) = OP(x, a_1, ..., a_n)$ where OP is some operation in the rewrite system and $a_1, .., a_n$ are terms.

   Thus, x is a 'sub-term' of the term $f(x)$.

   Let G' be $applyPred\ t'\ s'$. where t' and s' are some terms and $Pred\ s'\ t' \notin P_\Delta$.

   Thus, if $P_\Delta \vdash G'$ then $P_\Delta \vdash applyPred\ t\ s$ where t i some sub-term of t' and s is some sub-term of s'.

   By the inductive assumption: $\Delta \vdash (s = t)$.

   We can look at s as $f(s')$ and t as $f(t')$. Thus:

$$\text{CONG}\ \frac{\Delta \vdash s = t}{\Delta \vdash f(s) = f(t)}$$

   Thus, $P_\Delta \vdash G \Rightarrow \Delta \vdash s' = t'$

Thus, inductively proved that if $(P_\Delta \vdash G \Rightarrow G)$ then $(\Delta \vdash (s = t))$.

$\square$

Theorem 3. *Let G be onestep t Cert r. If $P_\Delta \vdash G$ then $\Delta \vdash r = t$.*

Proof. By induction on the prolog proof of $P_\Delta \vdash G$.

1. If $Cert$ is empty then G must be $onestep\ t\ Cert\ t$.

$$\text{REFLEX}\ \frac{}{\Delta \vdash t = t}$$

   Thus, $P_\Delta \vdash G \Rightarrow \Delta \vdash (s = t)$.

2. Let $P_\Delta \vdash G \Rightarrow \Delta \vdash (r = t)$.

   Let G' be $onestep\ t\ [Pred \mid Cert]\ r'$.

   Then, if $P_\Delta \vdash G'$ then:

   (a) $P_\Delta \vdash applyPred\ r'\ r$. Thus, by Theorem 2, $\Delta \vdash r' = r$.

   (b) $P_\Delta \vdash G$. Thus, by the inductive hypothesis $\Delta \vdash (r = t)$.

$$\text{TRANS}\ \frac{\Delta \vdash r' = r \qquad \Delta \vdash r = t}{\Delta \vdash r' = t}$$

   Thus, $P_\Delta \vdash G' \ \Rightarrow\ \Delta \vdash r' = t$.

Thus, if $P_\Delta \vdash G$ then $\Delta \vdash r = t$.

$\square$

```
verify(Tfinal, Certificate, Toriginal) :-  treeify(Tfinal, Rw),
treeify(Toriginal, T),
onestep(Rw, Certificate, T).
```

Theorem 4. *Let Goal be verify t c s. If $P \vdash$ Goal then $\Gamma \vdash (s = t)$.*

If $P \vdash$ *Goal*, then $P \vdash$ *treeify* t r and $P \vdash$ *treeify* s r' and $P \vdash$ *onestep* r c r'.

1. By the definition of P_Σ, if $P_\Sigma \vdash$ *treeify* s r', then $\Sigma \vdash (s = r')$.
2. By the definition of P_Σ, if $P_\Sigma \vdash$ *treeify* t r, then $\Sigma \vdash (t = r')$.
3. By Theorem 3, if $P_\Delta \vdash$ *onestep* r c r' then $\Delta \vdash r' = r$.

As $\Sigma \subseteq \Gamma$ and $\Delta \subseteq \Gamma$, and $P_\Sigma \subseteq P$ and $P_\Delta \subseteq \Delta$:

$$\cfrac{\cfrac{\Gamma \vdash (s = r') \qquad \Gamma \vdash (r' = r)}{\Gamma \vdash (s = r)}\,\text{Trans.} \qquad \cfrac{\Gamma \vdash (t = r)}{\Gamma \vdash (r = t)}\,\text{Sym}}{\Gamma \vdash (s = t)}\,\text{Trans}$$

Thus, given the relation between P and Γ, if $P \vdash$ *Goal* then $\Gamma \vdash (s = t)$.

6 Conclusion

Given a proof certificate p_c of a proof p, Theorem 3 and Theorem 4 prove that if the verifier decides that p_c is valid, then p must be correct. p is produced by the verifier program in the form of an internal Prolog proof [2], using only the Axiom, Instantiation, Symmetric, Transitivity and Congruence rules.

Thus, the system presented in this report is *certified* correct using formal mathematical methods. Any system that produces a proof certificate as described above is *certifiable*, as it can have individual rewrites certified by the system.

The verifier can help reduce the number of correctness checks in systems that involve rewriting, without sacrificing the validity of the results produced. Given the importance of rewriting and equational reasoning in automated deduction, symbolic algebra, programming languages, etc. it can help reduce redundancy and ensure correctness of several systems.

A Appendix

A.1 Inbuilt Predicates

Let A denote Prolog's inbuilt predicates.

1. Thm: If $A \vdash isList\ x$ then x is a list.

   Proof by Induction:

   Base Case: As [] is a list, $A \vdash isList\ [] \Rightarrow []$ is a list.

   Inductive case: Let $A \vdash isList\ x \Rightarrow x$ is a list.

   $A \vdash isList\ [y \mid x]$ if $A \vdash isList\ x$. Thus, x must be a list.

   Then, by the definition of a list, $[y \mid x]$ must be a list.

   Thus, if $A \vdash isList\ x$ then x is a list.

2. Thm:If $A \vdash notList\ x$ then x is not a list.

   By Prolog's definition, a list is defined as either an empty list, or a structure of the form $[Head \mid Tail]$. If $A \vdash notList\ x$ then x is of neither of these forms, and so cannot be a list.

3. Thm: If $A \vdash listmem\ x\ a\ n$ then x is the nth member of list a.

   Proof by Induction:

   Base case: If $A \vdash listmem\ 0\ [x \mid a]\ x$, then x is clearly the 0th member of list a.

   Inductive case: Let $A \vdash listmem\ n\ a\ x \Rightarrow x$ is the nth member of a.

   $A \vdash listmem\ n\ [b \mid a]\ x$ if $A \vdash listmem\ n\ a\ x$. Thus, x is the nth member of a. Then, by definition, x must be the $n+1$th member of list $[b \mid a]$.

   Thus, $A \vdash listmem\ n\ [b \mid a]\ x \Rightarrow x$ is the $n+1$th element of $[b \mid a]$.

   Hence proved.

4. Thm: If $A \vdash member\ x\ a\ n$ then x is the nth member of list a.

   Proof by Induction

   Base case: By the definition of a list, x is 0th member of the list $[x \mid Tail]$. Thus, $A \vdash member\ x\ [x \mid Tail]\ 0 \Rightarrow x$ is the 0th member of $[x \mid Tail]$.

   Inductive case: Let $A \vdash member\ x\ a\ n \Rightarrow x$ is the nth member of the list a.

   $A \vdash member\ x\ [b \mid a]\ (n+1)$ if $A \vdash member\ x\ a\ n$.

   Thus, by the inductive hypothesis, x is the nth member of of a. Thus, x must be the $n+1$th member of $[b \mid a]$.

   Thus, $A \vdash member\ x\ [b \mid a]\ (n+1) \Rightarrow x$ is the $n+1$th member of $[b \mid a]$.

   Hence proved.

5. Thm: If $A \vdash \textit{replace}\ n\ x\ b\ a$ then a is a copy of list b, except with x in its nth position.

   Proof by induction:

   Base case: If $A \vdash \textit{replace}\ 0\ [y \mid Tail]\ [x \mid Tail]$, then $[x \mid Tail]$ has x in position 0, and $[x \mid Tail]$ and $[y \mid Tail]$ are indentical from their 1st position on.

   Inductive case: (Inductive hyp) Let $A \vdash \textit{replace}\ n\ x\ b\ a \Rightarrow a$ is a copy of list b, except with x in its nth position.

   $A \vdash \textit{replace}\ n+1\ x\ [h \mid b]\ [h \mid a]$ if $A \vdash \textit{replace}\ n\ x\ b\ a$.

   Thus, by the inductive hypothesis, a is a copy of list b, except with x in its nth position. Thus, as $[h \mid b]$ and $[h \mid a]$ are identical in their 0th position, $[h \mid a]$ is a copy of $[h \mid b]$, except with x in its $n+1$th position.

   Thus, $A \vdash \textit{replace}\ n+1\ x\ [h \mid b]\ [h \mid a] \Rightarrow [h \mid a]$ is a copy of $[h \mid b]$, except with x in its $n+1$th position.

   Hence proved.

A.2 Representation of Terms and Rules

Terms are represented in the system as trees of the form $t(OP, L)$ where OP is an operation and L is a list of the proper subterms of the current term.

If a term is a constant or variable, it is of the form $t(X, [])$.

Thus, the subterms of a term are zero-indexed by integers, from left to right. Subterms in the Certificate List are identified by paths from the root that take the form of a list of integers.

For example, $(x+y)$ has path $[0, 1]$ in the term $t(*, [t(*, [t(z, []), t(+, [t(x, []), t(y, [])])]), t(w, [])])$.

Rewrite rules are specified as predicates in Prolog syntax. They are expressed in the form $< predicatename > (< initialterm >, < resultantterm >)$.

For example, the rule $1{*}X = X$ could be represented as $pred(t('*', [t(1, []), X]), X)$.

If the name of a rule is not explicitly mentioned in the 'Certificate List', it, by default, assumed to be 'pred'.

A.3 Example Certificate

This example proof certficate has the following rules:

1. $x * y) * z = x * (y * z)$
2. $a * -a = e$
3. $a * e = a$

It details a proof that $e = a * -((a * e) * (b * -b))$ using tacticals.

```
a * - ( (a*e) * (b * -b))
[((conv,inv),[]), ((conv, id),[1,0]), ((conv,inv), [1,0,1]), ((conv, id),[1,0,0])]
e

treeify((X * Y), t((*), [X_, Y_])) :- treeify(X, X_), treeify(Y, Y_).
treeify((- X), t((~), [X_])) :- treeify(X, X_).
treeify(X, t(X,[])).

assoc(t(('*'), [t(('*'), [X, Y]), Z]), t(('*'), [X, t(('*'), [Y, Z])])).
inv(t(*,[A, t(~, [A])]), t(e,[])).
id(t(*, [A, t(e, [])]), A).
```

References

[1] J. Harrison, "4.3 Equational Logic and Completeness Theorems", *Handbook of Practical Logic and Automated Reasoning* Cambridge University Press, 2009, pp 246-47.

[2] Miller, Dale, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. "Uniform Proofs as a Foundation for Logic Programming." Annals of Pure and Applied Logic 51.1-2 (1991): 125-57. Web.

[3] Birkhoff, Garrett, and P. Hall. "On the Structure of Abstract Algebras." Mathematical Proceedings of the Cambridge Philosophical Society 31.04 (1935): 433-54. Web.