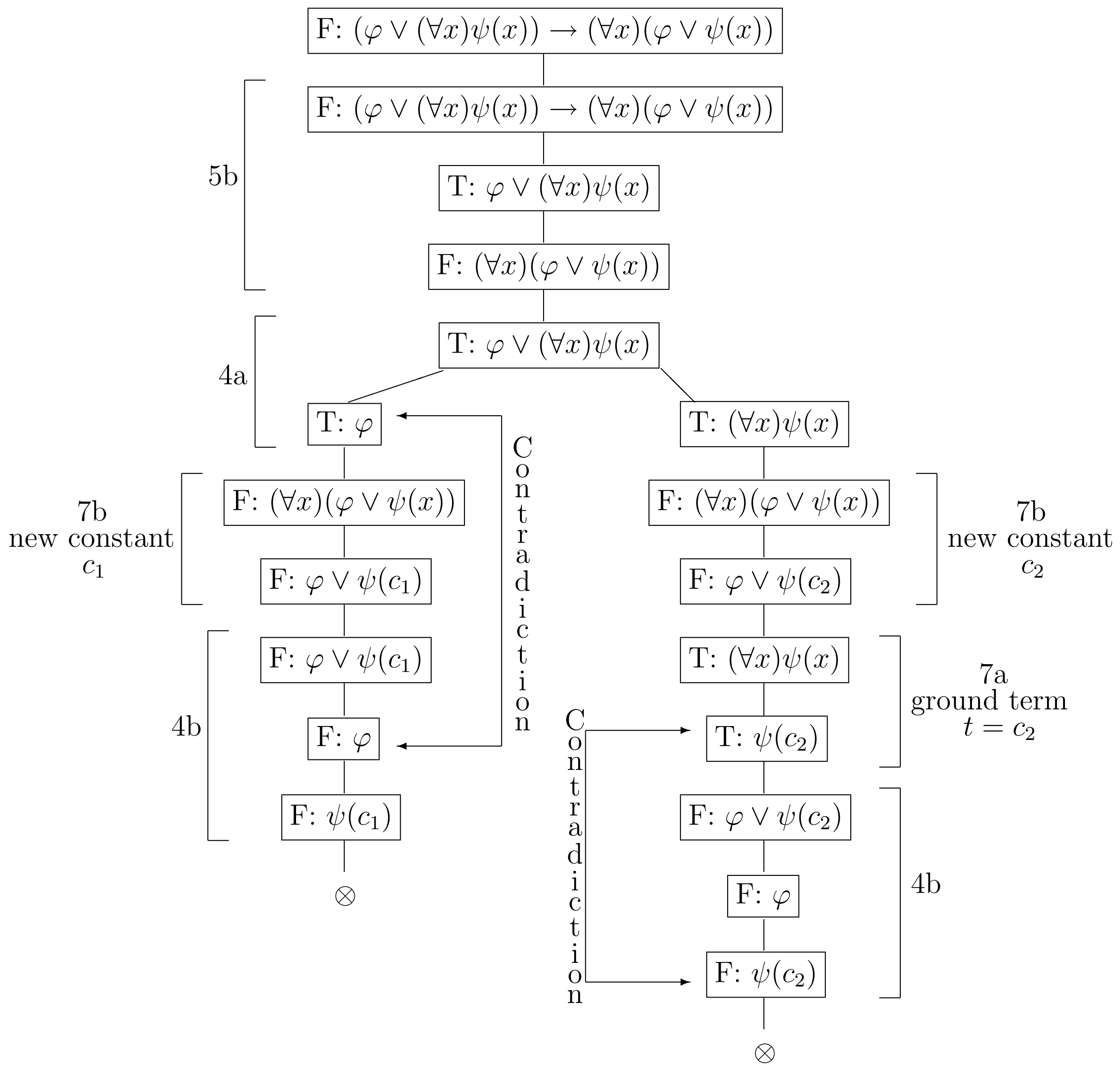
1 Tableau Proofs in First Order Logic

We will give a tableau proof of the following:

$$\text{“}(\varphi \vee (\forall x)\psi(x)) \to (\forall x)(\varphi \vee \psi(x)),\ x \text{ not free in } \varphi.\text{”}$$


F: $(\varphi \vee (\forall x)\psi(x)) \to (\forall x)(\varphi \vee \psi(x))$
F: $(\varphi \vee (\forall x)\psi(x)) \to (\forall x)(\varphi \vee \psi(x))$
5b
T: $\varphi \vee (\forall x)\psi(x)$
F: $(\forall x)(\varphi \vee \psi(x))$
T: $\varphi \vee (\forall x)\psi(x)$
4a
T: φ
T: $(\forall x)\psi(x)$
Contradiction
7b new constant c_1
F: $(\forall x)(\varphi \vee \psi(x))$
F: $\varphi \vee \psi(c_1)$
F: $(\forall x)(\varphi \vee \psi(x))$
F: $\varphi \vee \psi(c_2)$
7b new constant c_2
4b
F: $\varphi \vee \psi(c_1)$
F: φ
F: $\psi(c_1)$
$\otimes$
T: $(\forall x)\psi(x)$
T: $\psi(c_2)$
7a ground term $t = c_2$
Contradiction
F: $\varphi \vee \psi(c_2)$
F: φ
F: $\psi(c_2)$
4b
$\otimes$

Explaining the Tableau:

5b For an implication to be false, the hypothesis must be true and conclusion false.

4a If A or B is true, then either A is true or B is true (we have 2 alternatives).

7b If a predicate is false for all x, then it must be false for some particular x. We give this x a (new) name.

4b If A or B is false, then both A and B must be false.

7a If a predicate is true for all x, then it must be true for some particular x. We give this x a (new) name.

2 A Language with No Finite Model

Here is an example of a finite language $\mathcal{L}$ and a finite set of sentences S of $\mathcal{L}$ that has an infinite model but no finite model.

Let $\mathcal{L}$ be the language with one constant c, unary function f, and unary predicate P. Consider the (finite) set of sentences:

$$
\begin{aligned}
S = \ \{ \ & (\forall x)(\neg P(x,x)), \\
& (\forall x,y)(P(x,y) \rightarrow (\neg P(y,x))), \\
& (\forall x,y,z)((P(x,y) \wedge P(y,z)) \rightarrow P(x,z)), \\
& \forall x P(x, f(x)), \\
& (\forall x)(\exists y P(x,y)) \ \}
\end{aligned}
$$

A natural model for S is the structure $\mathcal{A}$ with domain $A = \mathbb{N} = \{0, 1, 2, ...\}$ the natural numbers, $c^{\mathcal{A}} = 0$, $f^{\mathcal{A}} : A \rightarrow A$ is the successor function defined by $f^{\mathcal{A}}(n) = n + 1$, and P is the "$<$" relation: $P^{\mathcal{A}} = \{(m,n) \in A \times A \,|\, m < n\}$.
Note: For the following discussion, let s and t be a ground terms with $s^{\mathcal{A}} = m$, $t^{\mathcal{A}} = n$.

To see that this is actually a model of S notice that since $n \not< n$, we have $(n,n) \notin P^{\mathcal{A}}$ thus $\mathcal{A} \vdash \neg P(t,t)$. Since t is arbitrary, $\mathcal{A} \vdash (\forall x)(\neg P(x,x))$.

We know that either $m < n$ or $n \leq m$ (not both). The former gives us $(m,n) \in P^{\mathcal{A}}$ and $(n,m) \notin P^{\mathcal{A}}$. Thus $\mathcal{A} \vdash \neg P(t,s)$. On the other hand, if $n \leq m$ we have that $(m,n) \notin P^{\mathcal{A}}$. Thus $\mathcal{A} \vdash \neg P(s,t)$. Putting these together we get $\mathcal{A} \vdash \neg P(s,t)$ or $\mathcal{A} \vdash \neg P(t,s)$. Hence, $\mathcal{A} \vdash P(s,t) \rightarrow (\neg P(t,s))$. Again since s and t are arbitrary, we conclude that $\mathcal{A} \vdash (\forall x,y)(P(x,y) \rightarrow (\neg P(y,x)))$.

Similarly, $\mathcal{A} \vdash (\forall x,y,z)((P(x,y) \wedge P(y,z)) \rightarrow P(x,z))$ holds because "$<$" is a transitive relation. Notice that $n < n+1 = f^{\mathcal{A}}(n)$. Thus $(n, f^{\mathcal{A}}(n)) \in P^{\mathcal{A}}$ thus $\mathcal{A} \vdash P(t, f(t))$. Therefore, $\mathcal{A} \vdash \forall x P(x, f(x))$. $\mathcal{A} \vdash P(t, f(t))$ implies that $\mathcal{A} \vdash \exists y P(t,y)$. Therefore, $\mathcal{A} \vdash (\forall x)(\exists y P(x,y))$. So we conclude, $\mathcal{A} \vdash S$ (S is satisfiable by an infinite model).

However, S has no finite model. Let $a_1 = c^{\mathcal{A}}$. By $(\forall x)(\exists y P(x,y))$ we know that there is some ground term t_2 (let $a_2 = t_2^{\mathcal{A}}$) such that $(a_1, a_2) \in P^{\mathcal{A}}$. a_2 is distinct from a_1 since otherwise we would have $(a_1, a_2) = (a_1, a_1) \in P^{\mathcal{A}}$, but we know $(\forall x)(\neg P(x,x))$.

Suppose that we have constructed a list $t_1, t_2, ..., t_l$ (let $a_j = t_j^{\mathcal{A}}$) such that each member of the list is distinct and $(a_j, a_l) \in P^{\mathcal{A}}$ for each $j = 1, ..., (l-1)$. Let's extend this list. Let $t_{l+1} = f(t_l)$ (and $a_{l+1} = t_{l+1}^{\mathcal{A}}$). We have $P(t_l, f(t_l)) = P(t_l, t_{l+1})$. But we also know that $P(t_j, t_l)$ for each $j = 1, ..., (l-1)$. Thus by $(\forall x,y,z)((P(x,y) \wedge P(y,z)) \rightarrow P(x,z))$ we conclude that $P(t_j, t_{l+1})$ for each $j = 1, ..., (l-1)$. Hence, $P(t_j, t_{l+1})$ for each $j = 1, ..., l$.

Suppose that we have already listed t_{l+1} say $t_{l+1} = t_j$ (some $j = 1, ..., l$). Then $P(t_j, t_{l+1}) = P(t_{l+1}, t_{l+1})$ but this cannot be since $(\forall x)(\neg P(x,x))$. Hence each element in our list is distinct.

So we can constuct an arbitrarily long list of distinct elements in any structure which models S. Therefore, S has no finite model.

3 Prenex Form and Skolemizations

The idea behind prenex form, is that one can "pull all the quantifiers out in front." Once a sentence is in prenex form, you can easily remove existential quantifiers by replacing the corresponding variables with "new names." Although we must be careful to state the dependence of these "names" on other variables. For example, $\exists x\, A(x)$ can be replaced by $A(f)$ where f is a constant. On the other hand, $\forall y \exists x\, A(x, y)$ can't be replaced by $\forall y\, A(f, y)$ where f is a constant because an x exists for each y – there is a dependence of x on y. So instead of a constant, we replace x with a function of y. That is $\forall y \exists x\, A(x, y)$ can be replaced by $\forall y\, A(f(y), y)$. So any sentence in prenex form can be "Skolemized" by replacing variables bound by an existential quantifier with functions.

Example: Find prenex equivalents and Skolemizations for the following sentences:

(a) $\forall y(\exists x P(x, y) \to Q(y, z)) \land \exists y(\forall x R(x, y) \lor Q(x, y))$.

(b) $\exists x R(x, y) \leftrightarrow \forall y P(x, y)$."

Part (a) I have added parentheses to make the meaning of this "sentence" clear:

$$(((\forall y)(((\exists x)P(x, y)) \to Q(y, z))) \land ((\exists y)((\forall x)(R(x, y) \lor Q(x, y)))))$$

Now apply the following rules to transform to prenex form [the labels come from a particular logic textbook].

(3a) $((\forall u)(((((\exists x)P(x, u)) \to Q(u, z))) \land ((\exists y)((\forall x)(R(x, y) \lor Q(x, y))))))$

(3b') $((\forall u)((\exists v)(((((\exists x)P(x, u)) \to Q(u, z))) \land ((\forall x)(R(x, v) \lor Q(x, v))))))$

(4b) $((\forall u)((\exists v)((((\forall w_0)(P(w_0, u) \to Q(u, z)))) \land ((\forall x)(R(x, v) \lor Q(x, v))))))$

(3a) $((\forall u)((\exists v)((\forall w)((P(w, u) \to Q(u, z)) \land ((\forall x)(R(x, v) \lor Q(x, v)))))))$

(3a') $((\forall u)((\exists v)((\forall w)((\forall x_0)((P(w, u) \to Q(u, z)) \land (R(x_0, v) \lor Q(x_0, v)))))))$

Finally, let us drop some of the parentheses and substitute x for x_0. This should make the formula easier to read. [Also, add a universal quantifier to bind z.]

$$\forall z \forall u \exists v \forall w \forall x((P(w, u) \to Q(u, z)) \land (R(x, v) \lor Q(x, v)))$$

To Skolemize we remove the existential quantifier "$\exists v$" and replace every occurance of v with $g(u, z)$.

$$\forall z \forall u \forall w \forall x((P(w, u) \to Q(u, z)) \land (R(x, g(u)) \lor Q(x, g(u, z))))$$

Part (b) Again, I have added parentheses to make the meaning of this "sentence" clear:

$$(((\exists x)R(x, y)) \leftrightarrow ((\forall y)P(x, y)))$$

Before pulling quantifiers out, we need to eliminate the connective "$\leftrightarrow$". The above sentence is equivalent to:

$$((((\exists x)R(x, y)) \to ((\forall y)P(x, y))) \land (((\forall y)P(x, y)) \to ((\exists x)R(x, y))))$$

Now apply the following rules:

(4b) $(((\forall u_0)(R(u_0,y) \to ((\forall y)P(x,y)))) \wedge (((\forall y)P(x,y)) \to ((\exists x)R(x,y))))$

(3a) $((\forall u)((R(u,y) \to ((\forall y)P(x,y))) \wedge (((\forall y)P(x,y)) \to ((\exists x)R(x,y)))))$

(4a') $((\forall u)(((\forall v_0)(R(u,y) \to P(x,v_0))) \wedge (((\forall y)P(x,y)) \to ((\exists x)R(x,y)))))$

(3a) $((\forall u)((\forall v)((R(u,y) \to P(x,v)) \wedge (((\forall y)P(x,y)) \to ((\exists x)R(x,y))))))$

(4a) $((\forall u)((\forall v)((R(u,y) \to P(x,v)) \wedge ((\exists w_0)(P(x,w_0) \to ((\exists x)R(x,y)))))))$

(3b') $((\forall u)((\forall v)((\exists w)((R(u,y) \to P(x,v)) \wedge (P(x,w) \to ((\exists x)R(x,y)))))))$

(4b') $((\forall u)((\forall v)((\exists w)((R(u,y) \to P(x,v)) \wedge ((\exists z_0)(P(x,w) \to R(z_0,y)))))))$

(3b') $((\forall u)((\forall v)((\exists w)((\exists z)((R(u,y) \to P(x,v)) \wedge (P(x,w) \to R(z,y)))))))$

Now let's drop some of the parentheses to make this formula easier to read.

$$\forall u \forall v \exists w \exists z((R(u,y) \to P(x,v)) \wedge (P(x,w) \to R(z,y)))$$

To Skolemize we need to introduce to new functions, one for w and one for z. Both of these functions will depend on u and v since the univerisal quantifiers $\forall u \forall v$ appear to the left of these existentials. Replace w with $f(u,v,x,y)$ and z with $g(u,v,x,y)$ and we get [Again, add universal quantifiers to bind the free variables x and y]:

$$\forall x \forall y \forall u \forall v((R(u,y) \to P(x,v)) \wedge (P(x,f(u,v)) \to R(g(u,v),y)))$$

4 Resolution Again: Unification

To apply resolution to predicates you must deal with variables. The process of replacing variables with other terms to make things match is called unification.

Example: Find substitutions that unify the following sets of expressions:

(a) $\{P(x,f(y),z), P(g(a),f(w),u), P(v,f(b),c)\}$

(b) $\{Q(h(x,y),w), Q(h(g(v),a),f(v)), Q(h(g(v),a),f(b))\}$.

Part (a) $\{P(x,f(y),z), P(g(a),f(w),u), P(v,f(b),c)\}$. Scanning through we must first match x and $g(a)$ and thus v and $g(a)$ also. This gives $\{P(g(a),f(y),z), P(g(a),f(w),u), P(g(a),f(b),c)\}$. The next difference is y and w. So we get $\{P(g(a),f(y),z), P(g(a),f(y),u), P(g(a),f(b),c)\}$. This leaves the mismatch of y and b, thus we get $\{P(g(a),f(b),z), P(g(a),f(b),u), P(g(a),f(b),c)\}$. Finally, z and u must be matched with c. Therefore, putting all this together we get the substitution:

$$\theta = \{x/g(a), v/g(a), y/b, w/b, u/c, z/c\}$$

Applying this substitution we get $\{P(g(a),f(b),c)\}$.

Part (b) $\{Q(h(x,y),w), Q(h(g(v),a),f(v)), Q(h(g(v),a),f(b))\}$. The first difference is x and $g(v)$. Matching these we get: $\{Q(h(g(v),y),w), Q(h(g(v),a),f(v)), Q(h(g(v),a),f(b))\}$. Next we need to match y and a. This gives us: $\{Q(h(g(v),a),w), Q(h(g(v),a),f(v)), Q(h(g(v),a),f(b))\}$. Now we need to match w and $f(v)$. We then have: $\{Q(h(g(v),a),f(v)), Q(h(g(v),a),f(v)), Q(h(g(v),a),f(b$
Finally, we match v with b. This substitution forces us to update the substitions $\{x/g(v)\}$ and $\{w/f(v)\}$ to $\{x/g(b)\}$ and $\{w/f(b)\}$. This gives the following substitution:

$$\theta = \{x/g(b), y/a, w/f(b), v/b\}$$

Applying this substitution we get $\{Q(h(g(b),a),f(b))\}$.

5 E-Mail in First Order Logic

We wish to construct a set of first-order, fully-quantified sentences that characterize the essential operation of an email system. We will make sure our formalization covers the following aspects of the email system:

- An email system involves messages, people, and hosts.
- All hosts are connected to each other in a network, but not necessarily directly connected to each other.
- The email system transmits each message from a source host to a destination host by successive steps across hosts on the network.
- People and hosts send and recieve messages at instants of time.
- Instants are ordered, and receipt of a message cannot occur earlier than emission of the message.
- All messages that are sent are received.

To begin we need to define "time" for our email system.

5.1 Time

Time involves two predicates, a unary predicate Time and a binary predicate After. $\text{Time}(t)$ is to be interpreted, "t is a moment in time." $\text{After}(t_1, t_2)$ is to be interpreted, "t_1 comes after t_2." The predicate After should only refer to moments in time so we include this axiom:

$$\forall t_1, t_2 \ \text{After}(t_1, t_2) \rightarrow \text{Time}(t_1) \wedge \text{Time}(t_2)$$

Moments in time are linearly ordered, so we need the following axioms:

- $\forall t_1, t_2, t_3 \ \text{After}(t_3, t_2) \wedge \text{After}(t_2, t_1) \rightarrow \text{After}(t_3, t_1)$. That is if time t_3 comes after t_2 which comes after t_1, then it follows that t_3 comes after t_1 (transitivity).

- $\forall t_1, t_2 \ \text{After}(t_2, t_1) \rightarrow \neg\text{After}(t_1, t_2)$. If t_2 is later than t_1, then it cannot be that t_1 is later than t_2 (anti-symmetry).

- $\forall t_1, t_2 \ \text{Time}(t_1) \wedge \text{Time}(t_2) \rightarrow \text{After}(t_1, t_2) \vee \text{After}(t_2, t_1) \vee (t_1 = t_2)$. Given two moments in time t_1 and t_2, either t_1 comes after t_2 or t_2 comes after t_1 or they are the same moment in time (time is totally ordered).

5.2 The Network

A network has hosts and connections between hosts. $\text{Host}(h)$ means that h is a host. $\text{Connection}(h_1, h_2)$ means that there is a direct connection linking h_1 to h_2. $\text{Connected}(h_1, h_2)$ mean that h_1 and h_2 are connected to each other in the network.

- $\forall h_1, h_2 \ \text{Connected}(h_1, h_2) \vee \text{Connection}(h_1, h_2) \rightarrow \text{Host}(h_1) \wedge \text{Host}(h_2)$. It only makes sense to say that h_1 is connected (or directly connected) to h_2 if both h_1 and h_2 are hosts.

- $\forall h_1, h_2$ Connection$(h_1, h_2) \leftrightarrow$ Connection(h_2, h_1). Direct connections must go both ways.

- $\forall h$ Host$(h) \rightarrow$ Connection(h, h). Every host is directly connected to itself.

- $\forall h_1, h_2$ Connection$(h_1, h_2) \rightarrow$ Connected(h_1, h_2). Any two directly connected hosts are also connected (directly connected implies connected).

- $\forall h_1, h_2, h_3$ Connected$(h_1, h_2) \wedge$ Connected$(h_2, h_3) \rightarrow$ Connected(h_1, h_3). If h_1 is connected to h_2 and h_2 is connected to h_3, then h_1 must be connected to h_3 (transitivity).

- $\forall h_1, h_2$ Host$(h_1) \wedge$ Host$(h_2) \rightarrow$ Connected(h_1, h_2). In our network, every host is connected to every other host. *Note*: This is not a *direct* connection.

5.3 People and Email Accounts

Let's introduce a two new unary predicates, Person and Account. Person(p) means that p is a person. Account(a) means that a is an email account. Every email account should have an owner and should exist on some host, so we introduce two functions, owner and hostedBy. owner(a) is the owner of account a and hostedBy(a) is the host which hosts the email account a.

- $\forall a$ Account$(a) \rightarrow$ Person(owner$(a)) \vee$ Host(owner(a)). This says that only people and hosts can own email accounts.

- $\forall a$ Account$(a) \rightarrow$ Host(hostedBy(a)). This says that email accounts are hosted by hosts.

- $\forall a$ Host(owner$(a)) \rightarrow$ (owner$(a) =$ hostedBy(a)). When a host owns an account, it should also host that account.

5.4 Messages

Message is a unary predicate. Message(m) means that m is an email message. Every message has a sender and a recipient, so we define to functions sender and recipient. sender(m) is the email account which sent the message m, and recipient(m) is the account to which the message was sent. A message is sent (and received) at some moment in time. Define two functions sent and received. sent(m) is the moment in time when the message m was sent and received(m) is the moment when the message was received. Finally, we also need to keep track of where the message goes between being sent and received, so we define a function of two variables location. location(m, t) is the host where m is residing at time t.

- $\forall m$ Message$(m) \rightarrow$ Account(sender$(m)) \wedge$ Account(recipient(m)). Messages are sent from email accounts to email accounts.

- $\forall m$ Message$(m) \rightarrow$ Time(sent$(m)) \wedge$ Time(received(m)). Messages are sent and received at moments in time.

- $\forall m$ Message$(m) \rightarrow$ (hostedBy(sender$(m)) =$ location$(m$, sent$(m)))$. Every message starts at the host of the sender's email account.

- $\forall m$ Message$(m) \rightarrow$ (hostedBy(recipient(m)) = location$(m,$ received(m))). Every message is delivered to the host of the recipient's email account at the time it is received.

- $\forall m, t$ Host(location(m, t)) $\leftrightarrow$ (After$(t,$ sent$(m)) \vee (t =$ sent$(m))) \wedge$ Message(m). If m is located on a host at time t, then m must be a message and t must be a time after the message was sent (or the time it was sent). Conversely, a message must reside on some host at every moment in time after it has been sent. As far as we are concerned, the message doesn't exist before it is sent. But once the message is sent, it exists from that time on.

- $\forall m, t$ Message$(m) \wedge$After$(t,$ received$(m)) \rightarrow$ (location$(m, t) =$ hostedBy(recipient(m)))). A message remains on its recipient's host once it has been received.

- $\forall m, t_1$ Host(location(m, t_1)) $\wedge \neg$(location$(m, t_1) =$ hostedBy(recipient(m)))) $\rightarrow$
  $\exists t_2$ After$(t_2, t_1) \wedge \neg$(location$(m, t_1) =$ location(m, t_2)) $\wedge$
  Connection(location(m, t_1), location(m, t_2)) $\wedge$
  ($\forall t_3$ After$(t_3, t_1) \wedge$ After$(t_2, t_3) \rightarrow$ location$(m, t_3) =$ location(m, t_1))).
  If at time t_1 a message still has not been delivered to its recipient's host, then there exists a future time t_2 when the message will be transferred to a different host which is directly connected to the present host. Moreover, for all times t_3 after t_1 and before t_2 the message will remain on the present host.

5.5 What's Missing?

There are many elements missing in this formalization. Like how is the next connection chosen? We just guarantee that one exists. Also, I have assumed that messages exist forever. What about deletion?

This is definitely an ideal email system in many ways since messages always get to their destination, hosts never go down, and all hosts are always connected. However, this formalization does capture the basics of an email system.