```
III: Typographical Number Theory
================================

In this section, we define Hofstadter's TNT [with some subtle modifications].

[TNT is PA]

Language of TNT
---------------

Alphabet:
    0 S + * ( ) =
    ~ /\ \/ =)
    a b c d e x y z '
    A E

(We no longer have propositional variables.)

Variables:
    a, b, c, d, e, x, y, z are variables
    If v is a variable, so is v'.

Terms:
    any variable is a term;
    0 is a term;
    if t and s are terms, then so are
        St,  (t+s),  (t*s);
    nothing else is a term.

wffs:
    if t and s are terms, then t=s is a wff (an _atomic formula_);
    if \phi and \psi are wffs, so are
        ~\phi,  <\phi /\ \psi>,  <\phi \/ \psi>,  <\phi =) \psi>;
    if \phi is a wff and v is a variable, then
        Av:\phi  and  Ev:\phi
    are wffs;
    nothing else is a wff.

Remark:
    We have unique parse trees.

Bound and free variables:
    An _occurrence of a variable_ v in a wff is a location in the wff where
    the variable appears, where appearances in substrings of the form "Av:" or
    "Ev:" do not count.
        (e.g. there are *no* occurrences of y in "Ay:y'=y'", but two of y')

    An occurrence of a variable v in a wff is _bound_ if it occurs within a
    substring of the form "Av:\phi" or "Ev:\phi" (\phi a wff). Else, the
    occurrence is _free_.

    The _free variables_ of a wff are those variables which occur free in the
    wff.

The standard interpretation:
    Variables stand for natural numbers.

    Call a choice of natural number for each variable, i.e. a map
    f : [variables] -> N, a "variable assignment".

    Given a variable assignment f, we evaluate terms as natural numbers:
        eval_f(v) = f(v)
        eval_f(0) = 0
        eval_f(St) = eval_f(t) + 1          ("Successor")
        eval_f((t+s)) = eval_f(t) + eval_f(s)
        eval_f((t*s)) = eval_f(t) * eval_f(s)

    Now we determine truth of a wff wrt a variable assignment f:

    * An atomic formula "t=s" is true wrt f iff eval_f(t) = eval_f(s).
    * "<\phi /\ \psi>" is true wrt f iff \phi and \psi are both true wrt f.
    * Similarly for ~, \/, =), as in propositional logic.
    * Av:\phi is true wrt f iff \phi is true for any
```

```
        variable assignment f' which is the same as f except possibly on v.
            (i.e. f'(w)=f(w) if w!=v)
        [ with notation:
            Av:\phi ^ f = T  iff  \forall f'. \forall w\in Variables.
                                  ((w!=v -> f'(w) = f(w)) -> \phi ^ f' = T)
        ]
    * Ev:\phi is true wrt f iff \phi is true for some
        variable assignment f' which is the same as f except possibly on v.


    Clearly, whether a wff \phi is true wrt a variable assignment f depends
    only on the values of f at the free variables of \phi.

    A wff with no free variables is a _sentence_, and is just true or false.

    A wff with 1 free variable expresses a _property_ of a natural number
        (e.g. primeness, oddness...).

    A wff with n free variables expresses an _n-ary relation_ (aka _predicate_)
        (1-ary == unary, 2-ary == binary etc)
        (e.g. "x is less than y"; "x is between y and z").

Examples:
    Ax: Ey: Sx=y
        (first think what Ey: Sx=y says about x
            (first think what Sx=y says about x,y))
    Ex: Ay: Sx=y
        (remark: cf ambiguity of english
            "every number is the predecessor of some number")
    Ax: Ey: x=Sy
    Ax: <Ey: x=Sy \/ x=0>

    Ey: (y+y)=x
    Ey: S(y+y)=x
    <Ey: (y+y)=x \/ S(y+y)=x>
    Ax: <Ey: (y+y)=x \/ S(y+y)=x>

    Ax: <Ey: (y+y)=x =) ~(x*x)=x>
    Ax: Ey: <(y+y)=x =) ~(x*x)=x>

    Ez: x=(y+z)
    Ax: Ez: (x*x)=(x+z)
    Ez: x=(y+Sz)
    Ax: Ez: (x*x)=(x+Sz)

    <~x=0 /\ <~x=S0 /\ Ay:Az:<(y*z)=x -> <y=x \/ z=x>>>>
    <Ey:x=SSy /\ ~Ey:Ez:x=(SSy*SSz)>

    Euclid:
        Ax: Ey: Ez: <y=(x+Sz) /\ ~Ey:Ez:x=(SSy*SSz)>

    Fermat (n=3):
        ~Ex:Ey:Ez:(x*(x*x))+(y*(y*y))=(z*(z*z))

    Goldbach:
        Ax:Ey:Ez:<<~Ey':Ez':y=(SSy'*SSz') /\ ~Ey':Ez':z=(SSy'*SSz')> /\ (y+z)=(x+x)>


Non-standard interpretations:
    A _structure in the language of arithmetic_ consists of
        a set N';
        an element 0' \in N';
        a unary function S' : N' -> N';
        binary functions +', *' : N'^2 -> N'.
        We denote the structure by <N';0',S',+',*'>, or just N'.

    The _standard_ arithmetic structure is <N;0,S,+,*>, i.e. the set of
        natural numbers N, with usual 0, successor, addition and
        multiplication.

    An assignment of variables for N' assigns an element _of N'_ to each
    variable; terms evaluate to elements of N' using 0', S', +', and *';
    wffs evaluate, wrt a variable assignment, to True/False as above (so "Ax:"
```

```
    now means "for all x in N'").

    For a sentence \sigma, we write
        N' |= \sigma
    to mean that \sigma is true when interpreted in N'.

    Example: the integers Z with usual zero, successor, addition, and
    multiplication is a structure in the language of arithmetic.
    "Ex:Sx=0" is true in Z but not in N.
        (Z |= Ex:Sx=0, but N |/= Ex:Sx=0)

PRED
----

Axioms:
    Axiom 0: Ax:x=x
Rules:
    Rules of the propositional calculus; premises of fantasies may now be
    arbitrary *TNT*-wffs.

    Generalisation: \phi |-> Av:\phi
        where v is a variable.
        RESTRICTION: v must not occur free in any premise of \phi.

    Specification: Av:\phi |-> \phi[t/v]
        where \phi[t/v] is the result of replacing each free occurrence of the
        variable v in \phi with the term t.
        RESTRICTION: any new occurrences of variables resulting from the
            substitution must be free.

    Interchange: XAv:~Y <-|-> X~Ev:Y
                 X~Av:Y <-|-> XEv:~Y
        where v is a variable;
        i.e. whenever "Av:~" occurs within a wff, it may be rewritten as
        "~Ev:", and vice-versa.

    Existence: \phi[t/v] |-> Ev:\phi
        where \phi is a wff, v is a variable, t is a term, and \phi[t/v] is
        the result of replacing each free occurrence of v in \phi with t.
        RESTRICTION: the substitution must meet the restriction imposed in the
            specification rule: any occurrences of variables created in
            passing from \phi to \phi[t/v] must be free.


    Symmetry: t = s |-> s = t
    Transitivity: (t = s, s = r) |-> t = r
    Congruence:
        t=s |-> St = Ss
        (t_1=s_1, t_2=s_2) |-> (t_1+t_2) = (s_1+s_2)
        (t_1=s_1, t_2=s_2) |-> (t_1*t_2) = (s_1*s_2)

        where t,s,r,t_i,s_i are terms.

Notation:
    |- \phi  means \phi is a PRED-theorem
        (note \phi can have free variables)

Examples:
    |- <Ax:Ay:x=y =) Ay:Sy=y>:
    [
        Ax:Ay:x=y
        Ay:Sy=y
    ]

    |- <Ax:Ay:x=y =) Ax:Sx=x>:
    [
        Ax:Ay:x=y
        Ay:Sy=y
        Sx=x
        Ax:Sx=x
    ]

    |- Ax:<Ey:<y=Sx /\ x=Sy> =) x=SSx>:
```

```
    [
        ~x=SSx
        [
            <y=Sx /\ x=Sy>
            x=Sy
            y=Sx
            Sy=SSx
            x=SSx
        ]
        <<y=Sx /\ x=Sy> =) x=SSx>
        <~x=SSx =) ~<y=Sx /\ x=Sy>>
        ~<y=Sx /\ x=Sy>
        Ay:~<y=Sx /\ x=Sy>
        ~Ey:<y=Sx /\ x=Sy>
    ]
    <~x=SSx =) ~Ey:<y=Sx /\ x=Sy>>
    <Ey:<y=Sx /\ x=Sy> =) x=SSx>
    Ax:<Ey:<y=Sx /\ x=Sy> =) x=SSx>
```

Non-examples (demonstrating necessity of the restrictions):

```
    [
        x=0
        Ax:x=0
    ]
    <x=0 =) Ax:x=0>
    Ax:<x=0 =) Ax:x=0>
    Uhoh!


    [
        Ax:Ey:~x=y
        Ey:~y=y
    ]
    <Ax:Ey:~x=y =) Ey:~y=y>
    Uhoh!


    Ax:x=x
    Ey:Ax:y=x
    Uhoh!


    [
        Ax:x=(x*S0)
        Ex:Ax:x=(x*x)
    ]
    Uhoh!


    [
        Ax:~x=Sx
        ~x=Sx
        Ex:~x=x     (existence, t:=Sx)
    ]
    Uhoh!
```

Definition:
    A _TNT-tautology_ is a substitution instance of a propositional tautology, obtained by replacing propositional variables with TNT-wffs.

Remark:
    By completeness of PROP, any TNT-tautology is a PRED-theorem.
    (Make the substitution in a PROP-derivation, yielding a PRED-derivation)

Example:

```
    |- <Ex:~x=x =) (SS0+SS0)=SSSSS0>:
    [
        Ex:~x=x
        ~Ax:x=x          (interchange)
        Ax:x=x           (axiom 0)
        <Ax:x=x /\ ~Ax:x=x>     (joining)
        [...lines proving following tautology omitted...]
```

<<Ax:x=x /\ ~Ax:x=x> =) (SS0+SS0)=SSSSS0>
(SS0+SS0)=SSSSS0 (detachment)
]
<Ex:~x=x =) (SS0+SS0)=SSSSS0> (fantasy rule)

For convenience, we add all TNT-tautologies to PRED as axioms.

[omitting this for now; might put it in later if it surviving without it is too annoying:
Lemma:
Suppose **|- <<\phi =) \phi'> /\ <\phi' =) \phi>>**,
and suppose **\theta** is a formula in which **\phi** occurs as a subformula, and **\theta'** is the result of replacing that subformula with **\phi'**.
Then **|- <<\theta =) \theta'> /\ <\theta' =) \theta>>.**
Proof:
[omitted, but straightforward by induction on length of **\theta**, and using the previous remark]

For convenience, we add as a rule to PRED:
Substitution: **\theta |-> \theta'**
whenever **\theta** and **\theta'** are as in the previous lemma.
]

Remark:
The existence rule can be deduced from the specification rule and interchange:
[
~Ev:\phi
Av:~\phi
~\phi[t/v]
]
<~Ev:\phi =) ~\phi[t/v]>
<\phi[t/v] =) Ev:\phi>

Soundness and completeness

Notation:
For **\Sigma** a set of sentences and **\tau** a sentence, write
\Sigma |- \tau
if **\tau** is a theorem of the system **PRED+\Sigma** obtained by adding **\Sigma** as axioms to PRED, and
\Sigma |= \tau
if **\tau** is satisfied by every structure in the language of arithmetic which satisfies all the sentences in **\Sigma**; i.e.
\Sigma |= \tau <=> for all **N'**: **N'|=\Sigma => N'|=\tau**

Fact (Soundness):
If **\Sigma |- \tau**, then **\Sigma |= \tau**

Fact (Gödel's Completeness Theorem):
If **\Sigma |= \tau**, then **\Sigma |- \tau**

(So **\Sigma |= \tau <=> \Sigma |- \tau**)

// Allaying possible confusion concerning the term 'complete':
Definition:
A system in the language of TNT is _complete for the standard **interpretation**_, abbreviated "**N**-complete" or "complete for **N**", if every TNT-sentence which is true in the standard interpretation is a theorem.

It is _negation complete_ if for every TNT-sentence **\sigma**, at least one of **\sigma** and **~\sigma** is a theorem.

Remark:
N-complete **=>** negation complete.

PRED is not negation complete!
e.g. neither **0+0=0** nor **~0+0=0** is a theorem!

GIT1 proves negation incompleteness, hence **N**-incompleteness.

TNT'

Definition:
TNT' is the system obtained by adding the following axioms to PRED:
Axiom 1: **Ax:~Sx=0**
Axiom 2: **Ax:(x+0)=x**
Axiom 3: **Ax:Ay:(x+Sy)=S(x+y)**
Axiom 4: **Ax:(x*0)=0**
Axiom 5: **Ax:Ay:(x*Sy)=((x*y)+x**)
Axiom 6: **Ax:Ay:<Sx=Sy =) x=y>**

We will also write "TNT'" to refer to the set of these 6 axioms, so
TNT' **|- \sigma**
means that **\sigma** is a TNT'-theorem.

Remark:
Hofstadter has the rule
Drop **S**: **Sx=Sy |-> x=y**
in place of Axiom 6; this makes no real difference - the two systems prove the same theorems.

[Remark for the initiated: TNT' is basically Robinson's **Q**, although we're missing the axiom that only 0 has no predecessor (which is ok for our purposes, as this is implied by the induction axioms)]

Note **N |=** TNT', so by soundness if TNT' **|- \sigma** then **N |= \sigma**.

Example:
TNT' **|- S0+S0=SS0**:

Ax:Ay:(x+Sy)=S(x+y)
Ay:(S0+Sy)=S(S0+y)
(**S0+S0)=S(S0+0)**
Ax:(x+0)=x
S0+0=S0
S(S0+0)=SS0
(**S0+S0)=SS0**

Fact:
TNT' can prove every sentence which is true in **N** of the form **t=s**, where **t** and **s** are terms.

Example:
TNT' **|- Ax:(x*(S0+S0))=((x*S0)+x**):

(**S0+S0)=SS0 (shown above)**
Ax:x=x
x=x
(**x*(S0+S0))=(x*SS0)**
Ax:Ay:(x*Sy)=((x*y)+x)
Ay:(x*Sy)=((x*y)+x)
(**x*SS0)=((x*S0)+x**)
(**x*(S0+S0))=((x*S0)+x**)
Ax:(x*(S0+S0))=((x*S0)+x)

TNT' is still not **N**-complete!

In other words, there are "non-standard" structures in the language of arithmetic which satisfy axioms 1-6, but satisfy sentences **N** does not.

Example:
Let **Mat_2(N)** be the structure in the language of arithmetic consisting of 2x2 matrices with natural number entries, with matrix addition and matrix multiplication and with **S(M) := M+I**, where I is the identity matrix.

Then **Mat_2(N) |=** TNT'.

Now let **\sigma** be the sentence **Ax:<x*x=0 =) x=0>**.

Clearly **N |= \sigma**.

But **M |= ~\sigma**, since

$$\begin{pmatrix}0&1\\0&0\end{pmatrix}^2 = \begin{pmatrix}0&0\\0&0\end{pmatrix}$$

So by soundness of PRED, {Ax 1−6} **|/− \sigma**.

However, by the Fact above, whenever **t** is a numeral
(i.e. one of 0, S0, SS0, ...),
<(t*t)=0 =) t=0>
is a TNT'−theorem!

Similarly, the following are ***not*** TNT'−theorems:
Ax:(0+x)=x
Ax:Ay:(x+y)=(y+x)
Ax:Ay:(x*y)=(y*x).

[Remark: **Mat_2(\N) |/= Q**, though]

TNT
===

What's missing?

In **N**, if **\phi[0/v]** and **\phi[S0/v]** and **\phi[SS0/v]** and so on
all hold, then so does **Av:\phi**.

But e.g. if **\phi := <(x*x)=0 =) x=0>**, then **\phi[0/x]** and **\phi[S0/x]** and
\phi[SS0/x] and so on are all theorems, but **Av:\phi** is not. Similarly with
\phi := (0+x)=x.

Proposed "Rule of All":
(**\phi[0/v]**, **\phi[S0/v]**, **\phi[SS0/v]**, ...) **|−> Av:\phi**

BUT rules of formal systems have ***finitely*** many inputs, this has ***infinitely***
many. You could never use this rule as part of a finite derivation!

Consider how we prove statements of the form "for all **n**" in everyday
mathematics...

Induction rule:
(**\phi[0/v]**, **Av:<\phi =) \phi[Sv/v]>**) **|−> Av:\phi**
where **v** is a variable and **\phi** is a wff, and **\phi[t/v]** is the result
of replacing each free occurence of **v** in **\phi** with the term **t**).

// But let's use axioms rather than a rule, so we have access to soundness and
// completeness.

Definition:
TNT is the system obtained by adding to TNT' the following infinite set of
axioms:
Induction axioms: for each wff **\phi** with one free variable **v**, the axiom
<<\phi[0/v] /\ Av:<\phi =) \phi[Sv/v]>> =) Av:\phi> .

(Again, we will also use "TNT" to refer to the set of axioms)

Remark:
TNT is more commonly known as PA ("**(first−order)** Peano arithmetic")

Example: **Ax:(0+x)=x** is a TNT−theorem:
1. **Ax:(x+0)=0**
2. (**0+0)=0**
3. [
4. **(0+x)=x**
5. **Ax:Ay:(x+Sy)=S(x+y)**
5. **Ay:(0+Sy)=S(0+y)** **(spec x−>0)**
6. **(0+Sx)=S(0+x)** **(spec y−>x)**
7. **S(0+x)=Sx**

8. (**0+Sx)=Sx**
9.]
10. **<(0+x)=x =) (0+Sx)=Sx>**
11. **Ax:<(0+x)=x =) (0+Sx)=Sx>** **(gen)**
12. **Ax:(0+x)=x** **(induction: lines 2, 11)**

Remark:
N |= TNT, so any TNT-theorem is true in **N (i.e. TNT is sound for N)**.

Question:
Does the converse hold? i.e. is TNT complete for **N**?

We will answer this presently!

Related question:
If a structure **N'=<N';S',+',*'>** satisfies TNT, must every element of **N'** be one of 0', S'0', S'S'0', ...?

Answer: no! However, there aren't any easily described examples like **Mat_2(N)**. [See Tennenbaum's theorem]

Fact:
There is a Post formal system, FormalTNT, such that a wff is a theorem of TNT iff it is a theorem of FormalTNT.

Appendices:
===========

A: deviations from Hofstadter

PRED is set up so as to satisfy Gödel's completeness theorem - this meant adding **Ax:x=x** as an axiom, and inserting the congruence rules. I also added the other form of interchange, because surviving without it is painful (though possible).

The existence rule got rewritten. Here's an equivalent version which looks more like Hofstadter's:

Existence: **\phi |-> Ev:\phi'**
where **\phi** is a wff, **v** is a variable, **t** is a term, and **\phi'** is the result of replacing one or more occurrences of **t** in **\phi** with **v**.
RESTRICTION: no bound occurrences of variables may be created or destroyed in passing from **\phi** to **\phi'**, and there may be no occurrences of **v** in **\phi'** other than those introduced through replacing occurrences of **t** in **\phi** with **v**.

"Drop **S**" became Axiom 6, and the induction rule became a set of axioms, to ensure that TNT is of the form **PRED+\Sigma**.

B: FormalTNT

We can implement TNT in a Post formal system.

It's more than a little ugly! But conceptually it's straight-forward.

[Again, I'm omitting this from the lectures, but including it here for the curious.]

(note that 'x', 'y' and 'z' are now in our alphabet, so we use 'X', 'Y', 'Z', 'X1', 'Z37' and so on for variables when giving production rules.

Let's simplify things by removing **E** from our formal system, considering "Ev:" to be just an abbreviation for "**~Av:~**".

Alphabet: as above, but add new symbols **|-** ? **|**, and all the roman alphabet in lower case and in upper case, except **X Y** and **Z**.
Axioms and Production rules:

Var|x
Var|y
Var|z
Var|X |-> Var|X'
Var|X |-> Term|X

```
VarNeq|x|y
VarNeq|y|z
VarNeq|z|x
VarNeq|X|Y |-> VarNeq|Y|X
(Var|X, Var|XY') |-> VarNeq|X|XY'

Term:0
Term|X |-> Term|SX
(Term|X, Term|Y) |-> Term|(X+Y)
(Term|X, Term|Y) |-> Term|(X*Y)

(Term|X, Term|Y) |-> WFF|X=Y

WFF|X   |->  WFF|~X
(WFF|X, WFF|Y) |-> WFF|<X/\Y>
(WFF|X, WFF|Y) |-> WFF|<X\/Y>
(WFF|X, WFF|Y) |-> WFF|<X=)Y>

// NoFree|Z|Y : variable Z doesn't appear free in wff Y
// NoFreeT|Z|Y : variable Z doesn't appear in term Y
VarNeq|Z|Y |-> NoFreeT|Z|Y
NoFreeT|Z|0
NoFreeT|Z|X |-> NoFreeT|Z|SX
(NoFreeT|Z|X, NoFreeT|Z|Y) |-> NoFreeT|Z|(X+Y)
(NoFreeT|Z|X, NoFreeT|Z|Y) |-> NoFreeT|Z|(X*Y)
(NoFreeT|Z|X, NoFreeT|Z|Y) |-> NoFree|Z|X=Y
(NoFree|Z|X, NoFree|Z|Y) |-> NoFree|Z|<X/\Y>
(NoFree|Z|X, NoFree|Z|Y) |-> NoFree|Z|<X\/Y>
(NoFree|Z|X, NoFree|Z|Y) |-> NoFree|Z|<X=)Y>
NoFree|Z|X |-> NoFree|Z|~X
(Var|Y, NoFree|Z|X) |-> NoFree|Z|AY:X
WFF|X |-> NoFree|Z|AZ:X

// NoFreePrems|Z|X : variable Z doesn't appear in any of the wffs in the
// ?-separated list X
Var|Z |-> NoFreePrems|Z|
(NoFree|Z|X, NoFreePrems|Z|Y) |-> NoFreePrems|Z|Y?X

(X|-Y, NoFreePrems|Z|X) |-> X|-AZ:Y     // (generalisation)

// Sub|X|Z|Z1|Y : Y is the result of validly substituting all free
// occurrences of the variable Z in the wff X with the term Z1, where
// "valid" means that no variable occurring in Z1 gets put in somewhere it
// gets bound.
// SubT|X|Z|Z1|Y : same, but X is a term.
(Var|Z, Term|Z1) |-> SubT|Z|Z|Z1|Z1
(Var|Z, Term|Z1) |-> SubT|0|Z|Z1|0
SubT|X|Z|Z1|Y |-> SubT|SX|Z|Z1|SY
(SubT|X|Z|Z1|Y, SubT|X1|Z|Z1|Y1) |-> SubT|(X+X1)|Z|Z1|(Y+Y1)
(SubT|X|Z|Z1|Y, SubT|X1|Z|Z1|Y1) |-> SubT|(X*X1)|Z|Z1|(Y*Y1)
(SubT|X|Z|Z1|Y, SubT|X1|Z|Z1|Y1) |-> Sub|X=X1|Z|Z1|Y=Y1
(Sub|X|Z|Z1|Y, Sub|X1|Z|Z1|Y1) |-> Sub|<X/\X1>|Z|Z1|<Y/\Y1>
(Sub|X|Z|Z1|Y, Sub|X1|Z|Z1|Y1) |-> Sub|<X\/X1>|Z|Z1|<Y\/Y1>
(Sub|X|Z|Z1|Y, Sub|X1|Z|Z1|Y1) |-> Sub|<X=)X1>|Z|Z1|<Y=)Y1>
Sub|X|Z|Z1|Y |-> Sub|~X|Z|Z1|~Y
(VarNeq|Z|Z2, NoFreeT|Z2|Z1, Sub|X|Z|Z1|Y) |-> Sub|AZ2:X|Z|Z1|AZ2:Y
(VarNeq|Z|Z2, Sub|X|Z|Z1|X) |-> Sub|AZ2:X|Z|Z1|AZ2:X
(Var:Z, WFF:X) |-> Sub|AZ:X|Z|Z1|AZ:X

(X|-AZ:Y, Term|Z1, Sub|Y|Z|Z1|Y1) |-> X|->Y1    // (specification)

(Term|X, Term|Y, Z|-X=Y) |-> Z|-Y=X (symmetry)
// other equality rules similar and omitted

// (the following is the same as for PROP)

|-
(X|-Y, WFF:Z)  |->  X?Z|-Z      // (pushing into a fantasy)
(X|-Y, WFF:Z)  |->  X?Z|-Y      // (carry-over)
(X?Y|-Z, WFF:Y) |->  X|-<Y=)Z>  // (popping out of a fantasy)

X|-<Y/\Z> |-> X|-Y
X|-<Y/\Z> |-> X|-Z
```

```
(X|−Y, X|−Z) |−> X|−<Y/\Z>
    // and so on for the other deduction rules of PROP

|−Ax:x=x
    // and similarly for axioms 1−6

// Induction axiom scheme:
(Sub|X|Z|0|Y, Sub|X|Z|SZ|Y1) |−> |−<<Y/\AZ:<X=)Y1>>=)AZ:X>

|−X  |−>  X                     // (deriving wffs)
```