I: Formal systems
=================

Definition:

An _alphabet_ is a finite set of _symbols_ (or _letters_ or _characters_).
e.g. {'a','b',...,'z'}

A _string_ (or _word_) in an alphabet \Sigma is a finite sequence of elements of \Sigma.
e.g. "aardvark", "word"

A _formal system_ on \Sigma comprises:
* a finite set of strings in the alphabet, called the _axioms_;
* a finite set of _production rules_.

A _derivation_ in a formal system is a finite sequence of strings (the _lines_ of the derivation) such that each line is an axiom or can be produced by a production rule from some preceding lines.

A string is a _theorem_ (or _production_) of a formal system if it is the last line of a derivation.

The _length_ of a derivation is the number of lines it has.

Before we define what production rules are, we must define patterns.
A _pattern_ in \Sigma is a string in the alphabet you get by adding to \Sigma some new symbols called _variables_ (as many of them as we need). We'll write these variables 'x', 'y', 'z', and use subscripts 'x_2' and so on if we need more.

So if the original alphabet \Sigma is { '-', 'p', 'q' }, then patterns are strings like "-xyp--x".

A _production rule_ comprises:
* A finite sequence of patterns in \Sigma, called the _inputs_;
* A single pattern, called the output. Each variable appearing in the output pattern must appear in at least one input pattern.

To define how production rules are applied, we should first define matching.

To _match_ a pattern to a string in \Sigma means to find strings in \Sigma which can substitute for the variables in the pattern so as to produce the string. e.g. "-xyp--x" matches "---qp----" by substituting "--" for "x" and "q" for "y".

To match a sequence of patterns to a sequence of strings means to match each pattern to the corresponding string, with the same substitutions being made when the same variable appears in more than one pattern. For example, ("-xyp--x","xy") matches ("---qp----","--q").

Finally, to apply a production rule to a sequence of strings means to match the input patterns to the strings, and produce as output the output pattern with variables substituted for strings according to the substitutions made in the matching. For example, the rule
("-xyp--x","xy") |-> "-yypx"
could be applied to the sequence of strings ("---qp----","--q"). to produce "-qqp--".

Note that sometimes there will be more than one way to match the given strings to the input patterns, resulting in different outputs. For example, the simple rule
"xxy" |-> "y"
when applied to the string "--p--pq-" could produce "p--pq-", but it could also produce "q-".

Remark:

This notion of formal system is due to Emil Post.
We will sometimes refer to them as "Post formal systems", when we want to be clear that we have this precise definition in mind.
The systems described in Hofstadter do not always fit rigidly into this

definition; in these notes, I aim to explain how they can be tweaked so as to do so.

The MIU-system

The MUI-system:

```
    Alphabet: {'M', 'I', 'U'}
    Axioms: {"MI"}
    Production rules:
        (I)     xI    |->  xIU
        (II)    Mx    |->  Mxx
        (III)   xIIIy  |->  xUy
        (IV)    xUUy   |->  xy
```

MU-puzzle: is "MU" a theorem?

Example: the following is a derivation in the MIU system:

```
    1.  MI
    2.  MIU         (produced by rule (I) from line 1)
    3.  MIUIU       (by (II) from 2)
    4.  MIUIUIUIUI  (by (II) from 3)
```

so "MIUIUIUIUI" is a theorem of the MIU system.

Example: the following is a derivation in the MIU system:

```
    MI
    MII
    MIIII
    MUI     (by (III) with x="M", y="I")
    MUIU
    MUIUUIU
    MUIIU
    MUIIUUIIU
    MUIIIIU
    MUIUU
```

Remark:
If we cut a derivation short, taking just the first **n** lines, what we have is also a derivation. So every line of a derivation is a theorem.

IU-puzzle: is "IU" an MIU-theorem?

Theorem: any MIU-theorem starts with 'M'
Proof by induction on the length of a derivation:
We show that for every natural number **k**

(*)_k every theorem with a derivation of length **<=k** starts with **M.**

(*)_0 is trivially true, as there are no theorems with derivations of length **<=0**!

Assume **(*)_k**, and consider a derivation of length **k+1.**

Each of the first **k** lines have derivations of length **<=k**, so they all start with 'M'.

The last line is an axiom or is produced from a previous line by one of **(I)-(IV)**.

If it is an axiom, it is "MI", which starts with 'M'.

If it was produced by (I) xI **|->** xIU:
"xI" starts with 'M', hence **x** does, hence "xIU" does.

Similar arguments apply for **(II)-(IV)**.

So **(*)_{k+1}** holds.

Example: The **MIU+** system is formed by adding a new production rule

(V) (MUx, MUy) **|->** MUxy

A derivation in this system:

1. MI
2. MII
3. MIIII
4. MUI
5. MUII (by (**V**) from (4) and (4))
6. MUIII (by (**V**) from (4) and (5))

Deciding theoremhood

Question: which strings in {'M', 'I', 'U'} are MIU-theorems?

First answer: those for which there exist derivations.

This is unsatisfactory!

We would like a _decision procedure_ for theoremhood:
a **procedure/algorithm/program** which we can carry out on any string, and which will (eventually) stop and give us an answer "yes" or "no", and which answers "yes" iff the string is a theorem.

We have "half" of that:
given a string, we can run through all possible derivations in order of length (see below), and stop with answer "yes" if the last line is equal to the given string.

This is a _semi-decision procedure_ for theoremhood:
an algorithm which, given a string **S**, answers "yes" if **S** is a theorem, but needn't stop at all if **S** isn't a theorem!

Algorithm to produce all derivations of a formal system, in order of length:
The only derivation of length 0 is the empty derivation.
Suppose we have produced all derivations of length **k**. To produce all derivations of length **k+1**:
- For each axiom and each length **k** derivation:
  append the axiom to the derivation, giving a length **k+1** derivation.
- For each production rule and each length **k** derivation:
  Say the production rule takes **n** strings as input.
  Run through each set of **n** lines from the derivation, and all the (finitely many!) ways to apply the production rule to them (choices for substitutions of variables). In each case, append the output, giving a length **k+1** derivation.

Remark:
For this argument to work, it's crucial that there be only finitely many axioms and finitely many production rules.

Remark:
If we remove rules (III) and (IV) of the MIU-system, we have an easy decision procedure: each rule increases the length of a string it acts on, so if a string **S** is a theorem it has a derivation of length at most the length of **S**. So just check all those derivations.

Why do we call the above procedure a "semi"-decision procedure?
Suppose we find a formal system Anti-MIU whose theorems are precisely the non-theorems of the MIU-system. Then we would have a decision procedure for MIU-theoremhood:
Given a string **S**, ***simultaneously*** run our semi-decision procedures for MIU and for Anti-MIU.
The first stops and says "yes" if **S** is an MIU-theorem;
the second stops and says "yes" if **S** is an Anti-MIU-theorem, i.e. if **S** is ***not*** an MIU-theorem.
So precisely one of them will eventually stop and say "yes"!
Then we stop, and say "yes" or "no" appropriately.

We'll come back to this idea later.

Solution to the MU-puzzle

Definition: For an MIU-string **S**, let **I(S)** be the number of occurences of 'I'.

Theorem: If **S** is an MIU-theorem, then **I(S)** is not divisible by 3
    (i.e. **I(S) != 0** mod 3)
Proof:
    By induction on length of derivations.
    Suppose **I(S') !=~ 0** mod 3 for any theorem **S'** having a derivation of
        length **<= k**, and suppose **S** has a derivation of length **k+1**.
    If **S** is an axiom, **S=**"MI" so 1 **= I(S) !=~ 0** mod 3.
    Else, **S** is produced by one of (**I)-(IV)** from some **S'** with **I(S') !=~ 0** mod 3.
        (I): **I(S) = I(S')**.
        (II): **I(S) = 2I(S')**, so **I(S) =~ 2I(S') !=~ 0** mod 3.
        (III): **I(S) = I(S')-3**, so **I(S) =~ I(S') !=~ 0** mod 3.
        (IV) **I(S) = I(S')**.
    So **I(S) !=~ 0** mod 3.

See assignment 1 for the converse.

Semantics

The pq-system:
    Alphabet: {'p', 'q', '-'}
    Axioms: {"-p-q--"}
    Production Rules:
        (I) xpyqz **|->** xpy-qz-
        (II) xpyqz **|->** **x**-pyqz-

Producing some theorems, it looks like every theorem is of the form
    "**-^np-^mq-^{n+m}**" (where "**-^n**" abbreviates **n** dashes).
So it's tempting to ***read*** **e**.g. "---p--q-----" as "3 plus 2 equals 5".

Is that what it "really means"?
Is "---p--q-----" ***true***, and "--p--q-----" ***false****?
What about "qpqpqq-"?

Definition:
    A _language_ in an alphabet is a set of strings, called the _well-formed
        strings_ (wfss).
    An _interpretation_ of a language is a way to assign a truth value (True
        or False) to each wfs.

So here, we're suggesting a language where the wfss are "**-^np-^mq-^k**" with
    n,m,k>=1, and the plus-equals interpretation:
        "**p**" **-->** "plus"
        "**q**" **-->** "equals"
        "-" **-->** "one"
        "--" **-->** "two"
        "---" **-->** "three"
        etc;
    so e.g. we assign True to "--p---q-----" because "three plus two equals
        five" is true.

We were led to this interpretation by noting that all theorems appeared to be
    true according to it.

Definition: **A** formal system is _consistent_ (or **_sound_**) with respect to an
    interpretation if all its theorems are wfss and are true under the
    interpretation.

Theorem:
    The pq-system is sound wrt the plus-equals interpretation.
Proof:
    The axiom "-p-q--" is true, since **1+1=2.**
    The production rule (I) preserves truth of wffs:
        if "**-^np-^m-^k**" is true, then **k=n+m**,
        so "**-^np-^m-q-^k**-" **=** "**-^np-^{m+1}q-^{k+1}**" is true,
            since **k+1 = (n+m)+1 = n+(m+1).**
    Similarly, so does (II).
    So (by an induction on length of derivations) every theorem is true.

Caution: "two plus three plus one equals six" makes sense, but
    "--p---p-q------" is ***not*** well-formed!

Remark:
Consider
"**p**" **-->** "equals"
"**q**" **-->** "subtracted from"
"-" **-->** "one"
etc.
This gives wfss the same truth values as the plus-equals interpretation.
Does that mean it's the *same* interpretation? This question is of no importance to us, and we will not give an answer.

Remark:
Consider the **plus-at_least** intepretation:
"**p**" **-->** "plus"
"**q**" **-->** "at least"
"-" **-->** "one"
etc.
The pq-system is also consistent wrt this interpretation!
But...

Definition:
A system is _complete_ with respect to an interpretation if every wfs which is true according to the interpretation is a theorem of the system.

Example: The pq-system is *not* complete with respect to the **plus-at_least** interpretation. Indeed, "-p-q-" is clearly not a theorem.

Theorem: The pq-system *is* complete wrt the plus-equals interpretation.
Proof:
We want to show that for any **n,m>=1**, **-^np-^mq-^{n+m}** is a theorem.
But indeed, **n-1** applications of (I) starting with the axiom yields "**-^np-q-^{n+1}**", and then **m-1** applications of (II) yields "**-^np-^m-^{n+m}**".

So the pq-system "captures" addition of two positive numbers.

More arithmetic in formal systems

Big question:
Can we find a language which we can interpret as making interesting statements in mathematics, and a complete consistent formal system for it?

Examples of the kinds of "interesting statements" we might want to express:
2+2 = 5 (we've got this covered, thanks to the pq-system!)
3*7 = 21
4^3 = 64
2+2 != 5
2*3 = 7 or 2 *** 3 = 6**
1337 is prime
For any integer **n**, **n*1 = n**
Every even number is the sum of two primes

Let's see what we can do!

The tq-system:
Alphabet: **{t,q}**
Axiom: -t-q-
Rules:
(I) xt-qz **|->** -xt-qz-
(II) xtyqz **|->** xty-qzx

Language: **-^nt-^mq-^k**
Interpretation: **-^nt^mq-^k --> n*t=k**

Soundness:
The axiom is true (**1*1=1**)
(I) and (II) preserve truth:
(I): **n*1 = m => (n+1)*1 = m+1**
(II): **n*m = k => n*(m+1) = k+n**
Completeness:
If **n*m=k**, we derive **-^nt-^mq-^k** from **-^nt-q-^n** by applying (II) **m-1** times, and we derive **-^nt-q-^n** from the axiom -t-q- by applying (I) **n-1** times.

```
    So the tq-system "captures" multiplication of two positive integers.

Compositeness:
    Add to the tq system a character 'C' and a rule of inference
            xqy |-> Cy
        and interpret C-^n as "n is composite".

    Completeness and soundness are easily checked.

Primeness:
    Can we find a system where P-^n is a theorem iff n is prime,
        i.e. iff n is *not* composite?
    The system for compositeness is no use to us here!
        (cf trying to find an anti-MIU system given only the MIU system...)
    We have to develop a new system.

    First, we capture "n does not divide m":
        Axioms: --DND-
        Rules:
            xDND-    |->  x-DND-
            xyDNDx   |->  xy-DNDx-
            xDNDy    |->  xDNDxy
        Interpretation:
            -^nDND-^m  --->  n does not divide m
                (i.e. m !=~ 0 mod n)

        (the first two rules give -^nDND-^m as a theorem whenever n>m)

    Secondly, we capture "n has no divisors among 2,3,...,m", which we can
        phrase as "n is Divisor Free up to m". Add the rules
            --DNDx  |->  xDF--
            (yDFx, x-DNDy)  |-> yDFx-

    Finally, add a rule:
            x-DFx  |->  Px-
        and an axiom:
            P--
```