② Sps $S = ((p \Rightarrow \neg q) \Rightarrow (p \vee (\neg r \wedge s)))$

Alg: $((x \Rightarrow \neg x) \Rightarrow (x \vee (\neg x \wedge x))) = S_0$
$((x \Rightarrow x) \Rightarrow (x \vee (\neg x \wedge x))) = S_1$
$(x \Rightarrow (x \vee (\neg x \wedge x))) = S_2$
$(x \Rightarrow (x \vee (x \wedge x))) = S_3$
$(x \Rightarrow (x \vee x)) = S_4$
$(x \Rightarrow x) = S_5$
$x = S_6 = S_7$ STOP

Alg says: S is a wff.

Correctness of recognition algorithm

Prop'n Sps S is a string and n_0 is least s.t. $S'_{n_0} = S_{n_0}$.
Then $S_{n_0} = x$ iff S is a wff.

Pf: ① ($\Leftarrow$) If S is wff, then $S_{n_0} = x$
Pf: Fact 1: $S(x)$ is wff
Pf: induction on construction of S (you try!)

Fact 2: Sps β is a wff whose only variable is x. Then:
@ $\beta = x$, or β contains a substring of the form $\neg x$ or $(x * x)$ for $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$

(2) ($\Rightarrow$) If $S_{n_0} = x$, S is wff

Pf:

Fact (i): if $S(x)$ is wff, then S is wff

Pf: induction (you try)

Fact (ii): if β is wff whose only var. is x, and we replace an occurrence of x w/ one of $\neg x$, or $(x * x)$, to form β', then β' is wff

Pf: induction.

- Now, to prove (2): Sps for our string S the algorithm yields

$$S(x) = S_0, S_1, \ldots, S_{n_0} = x$$

- for all $k < n_0$, S_k is obtained by replacing an occurrence of x in S_{k+1} by one of $\neg x$ or $(x * x)$

- hence by Fact (ii) each S_k is wff
- in particular $S_0 = S(x)$ is wff so by Fact (i), S is wff ✓

Polish notation

- we explore alternate ways of writing wff's

Defn. The Polish wffs or P-wffs are defined recursively as follows:

(i) every var. p is a P-wff
(ii) if A is a P-wff, so is $\neg A$
(iii) if A, B are P-wffs, so is $*AB$ for $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$

- P-wffs are written in lang. of PL w/o parentheses.
- just like regular wffs, P-wffs have parsing sequences.

Ex. $\Rightarrow\Rightarrow s\neg t\neg\vee p\wedge\neg qs$ is a P-wff

- a parsing sequence:

$s, q, \neg q, \wedge\neg qs, p, \vee p\wedge\neg qs, \neg\vee p\wedge\neg qs, t, \neg t, \Rightarrow s\neg t, \Rightarrow\Rightarrow s\neg t\neg\vee p\wedge\neg qs$

- this can be translated into a regular parsing sequence:

$s, q, \neg q, (\neg q \wedge s), p, (p \vee (\neg q \wedge s)),$
$\neg(p \vee (\neg q \wedge s)), t, \neg t, (s \Rightarrow \neg t),$
$((s \Rightarrow \neg t) \Rightarrow \neg(p \vee (\neg q \wedge s)))$

- so ↗ is corresponding wff

- despite losing parentheses we still have unique readability for P-wffs

Thm Every P-wff is in exactly one of the following forms:
(i) p, for a unique var. p
(ii) $\neg B$, for a unique P-wff B
(iii) $*BC$, for unique P-wffs B, C and a unique $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$

Pf. - only hard part is (iii)
- need to check if $*BC = \circ B'C'$ then $* = \circ$ (clear) and $B = B'$, $C = C'$
- will follow from:

Lemma No strict initial seg. of a P-wff is P-wff.
Pf: harder than for regular wffs!

Def'n we define a weight function w on strings:

$w(p) = 1$ for any var. p

$w(\neg) = 0$

$w(*) = -1$ for $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$

and for $S = s_1 s_2 \cdots s_n$ a string, define $w(S) = w(s_1) + \cdots + w(s_n) = \sum_{i=1}^{n} w(s_i)$

- e.g. $w(\wedge pq) = 1$ and $w(\wedge \Rightarrow \neg) = -2$

Fact For any P-wff A, have $w(A) = 1$

PF: induction (you try!)

Def'n (i) a terminal segment of a string $S = s_1 s_2 \cdots s_n$ is any string of the form $s_m s_{m+1} \cdots s_n$ for $1 \le m \le n$, or the empty string $\emptyset$.

(ii) For strings $S_1, S_2, \ldots, S_k$, their concatenation is the string $S_1 S_2 \cdots S_k$

Claim Any nonempty terminal segment of a P-wff A is a concatenation of P-wffs.

PF: induction on A:

(i) if $A = p$, clear

(ii) if $A = \neg B$ for a P-wff B all of whose terminal segments are concs of P-wffs, then if C is a terminal segment of A, either

C = A or C a term'l seg of B
- in either case (by induction)
C is a conc'n of P-wff's
(iii) if A = *BC for P-wff's
B, C whose final segments are
conc'ns of P-wffs
then any term'l seg D of A
either = A or is term'l seg of BC
in first case, done
in second, D either of form
B'C for B' a conc'n of P-wffs
or C' for C' a conc'n of P-wffs.
done in either case ✓

Ex Consider the P-wff
$\Rightarrow\Rightarrow s \neg t \neg \vee p \wedge \neg q s$ from above
terminal seg
= conc'n of $s, \neg t, \neg \vee p \wedge \neg q s$
all of which are P-wffs.

Pf of Lemma:

- Sps A is a P-wff and (for a contradiction) B is a strict init seg of A that is P-wff
- then A = BC for $C = C_1 C_2 \cdots C_n$

a conc'n of P-wffs (by claim)

- then $w(A) = w(B) + w(C)$
  $= w(B) + w(C_1) + \cdots + w(C_n)$
  $= 1 + (+ \cdots + 1) > 1$
- contradiction by our fact ✓

- unique readability follows:
  if $BC = B'C'$ and $B \neq B'$
  then w.l.o.g. B strict initial in B',
  contradiction ✓

Reverse Polish notation

- if we reverse conventions for forming P-wffs, get Reverse Polish notation

Def'n The RP-wff's are defined by:
(i) every var. p is RP-wff
(ii) if A is RP-wff, so is $A\neg$
(iii) if B, C are RP-wff's so is $BC*$ for $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$.

- by analogous proof for P-wffs have unique readability for RP-wff's

Ex $st\neg\Rightarrow pq\neg s\wedge v\neg\Rightarrow$
is an RP-wff w/ parsing sequence

$s, t, t\neg, st\neg\Rightarrow, p, q, q\neg,$
$q\neg s\wedge, pq\neg s\wedge v, pq\neg s\wedge v\neg,$
$st\neg\Rightarrow pq\neg s\wedge v\neg\Rightarrow$

- corresponds to our P-wff ex. above.

Translating RP-wffs to wffs

- we describe by way of example an algorithm for turning an RP-wff into a wff.

- consider
  $st\neg\Rightarrow pq\neg s\wedge v\neg\Rightarrow$
- find leftmost connective: here is $\neg$
- flip $t\neg$ to $\neg t$:
  $s\neg t\Rightarrow pq\neg s\wedge v\neg\Rightarrow$

- find next leftmost connective: here is $\Rightarrow$
- replace $s\neg t\Rightarrow$ with $(s\Rightarrow\neg t)$:
  $(s\Rightarrow\neg t)\, pq\neg s\wedge v\neg\Rightarrow$

- And next leftmost connective: ¬ again
- rpl. q¬ w/ ¬q:
  $(S \Rightarrow \neg t)\ p\ \neg q\ S \wedge \vee \neg \Rightarrow$
- keep going until done w/ all connectives:
  $(S \Rightarrow \neg t)\ p\ (\neg q \wedge S)\ \vee \neg \Rightarrow$
  $(S \Rightarrow \neg t)\ (p \vee (\neg q \wedge S))\ \neg \Rightarrow$
  $(S \Rightarrow \neg t)\ \neg (p \vee (\neg q \wedge S))\ \Rightarrow$
  $((S \Rightarrow \neg t) \Rightarrow \neg (p \vee (\neg q \wedge S)))$ ✓

- if A is an RP-wff, let O(A) denote string obtained by this alg.
- Observe: O(A) satisfies following recursive defn (which corresponds to last step of alg.):
  (i) $O(p) = p$ for a variable p
  (ii) $O(A\neg) = \neg O(A)$
  (iii) $O(AB*) = (O(A) * O(B))$
  for * any of $\wedge, \vee, \Rightarrow, \Leftrightarrow$

- by unique readability (of both wffs and RP-wffs) follows O defines a bijective correspondence between RP-wffs and wffs

- Inversely, can define an operation RP on the wffs by:

$RP(p) = p$

$RP(\neg A) = A\neg$

$RP((A * B)) = AB*$

- then not hard to see:

$RP(c(A)) = A$ for any RP-wff A

$c(RP(B)) = B$ for any wff B.

Conventions for abbreviating wffs:

- though P and RP notation more efficient we'll stick to our original way of writing wffs
- will use abbreviations sometimes by dropping parentheses:

Convention: ① $A * B$ abbreviates $(A * B)$

② the binding order for the binary connectives is $\wedge, \vee, \Rightarrow, \Leftrightarrow$

so e.g. $p \wedge q \vee r$ means $(p \wedge q) \vee r$

whereas $p \Rightarrow q \vee r$ means $p \Rightarrow (q \vee r)$