WFF 'N PROOF Guide and Workbook, Elementary

Revision .96

Table of Contents

Introduction

WFF 'N PROOF is a game based upon proving theorems in *propositional logic*[1]. The game requires math skills that are most typically taught in high school geometry classes, although the specific nature of proofs taught in geometry is different than what we learn in WFF 'N PROOF. Students learn the basics of rigorous mathematical reasoning.

This guide explains the fundamentals of WFF 'N PROOF, with particular emphasis on how to construct proofs. We start with an explanation of what exactly constitutes a well-formed formula (*wff*), and proof, then proceed to discuss the rules that are required for elementary division WFF 'N PROOF (i.e., *basic* WFF 'N PROOF).

Please consult the AGLOA website (www.agloa.org) for the latest WFF 'N PROOF rules. This guide discusses game rules only minimally, but it does include a section on *nonessentiality*, which is a topic that many students do not understand.

If you are interested in learning more about WFF ‘N PROOF, I strongly suggest reading Layman E. Allen’s original WFF ‘N PROOF rulebook and manual, written in 1962. This workbook uses some notational conventions from that Allen’s book.

Many thanks to Swami Jataja who provided valuable comments and suggestions.

1 Or, if one wishes to sound even more pompous, one could say *propositional calculus*.

Some Definitions

We use 4 different kinds of cubes in WFF 'N PROOF. They are

Sentence variables

These are the cubes **p**, **q**, **r**,and **s**.

Connectives

These are the cubes **C**, **A**, **K**, and **E**.

Negation

This is the cube **N**.

Rules cubes

These are the cubes **R**, **o**, and **i**.

What is a Wff?

We create sequences of cubes that we call *well-formed formulas*, or *wffs*. There are only 3 rules for determining if a sequence of cubes is a wff. A wff is

1. A sentence variable (p, q, r, s) all by itself, or
2. The negation cube (N) followed by a single wff, or
3. A connective (C, A, K, or E) followed by 2 wffs

For those wffs defined by rule 3, we say that the connective is followed by *wff1* and *wff2*. Furthermore, as we shall soon see, it is very helpful when learning WFF 'N PROOF to separate the connective, wff1, and wff2 with a "-".

Wff Examples

Some of the expressions below are wffs, and some are not.

Expression	Wff?	Reason
`p`	Yes	Rule 1
`Kpq`	Yes	`K-p-q`
`pq`	No	2 wffs, not 1
`KKpqq`	Yes	`K-Kpq-q`
`NErs`	Yes	`N-Ers`
`KqKpq`	Yes	`K-q-Kpq`
`Cp`	No	C followed by only 1 wff
`pNq`	No	Multi letter, does not start with Connective
`ApNpqNq`	No	Connective is followed by 4 wffs
`ApNp`	Yes	`A-p-Np`
`Nq`	Yes	`N-q`
`NNq`	Yes	`N-Nq`
`NNNq`	Yes	`N-NNq`
`KApNpNNNq`	Yes	`K-ApNp-NNNq`
`AAppp`	Yes	`A-App-p`
`AAAppppp`	Yes	`A-AAppp-p`
`i`	No	Not a sentence variable

Wff Worksheet 1

Each of the expressions below is a wff that begins with a connective. Rewrite each in “wff1 – wwf2” format. The first few have been done as examples.

Kpq	K-p-q	AANqrr	A-ANqr-r
ArNr	A-r-Nr	ANrr	A-Nr-r
ArKpr	A-r-Kpr	AErsCss	A-Ers-Css
KNpq	________	ENrKsp	________
KCpqErs	________	ECpqs	________
KCpqr	________	EqKrs	________
KsErs	________	EKrsq	________
KKpsr	________	CKqpr	________
CKKpsrr	________	CKrsNq	________
KEpNqr	________	KCKpqrq	________
EKpqr	________	KqCqAqp	________
EKrsNr	________	CqAqp	________
ACpqErs	________	KKrsApq	________
ArKCpqr	________	EKpqs	________
AKCpqrr	________	KNqCpr	________
AKprCqq	________	ANCprq	________
EsKpq	________	KCprNq	________
KArsp	________	ANAqpr	________
CKArspq	________	CKpqArq	________
CqKArsp	________	ANqNq	________

Wff Worksheet 2

1. Write a wff that uses exactly 1 cube.

2. Write a wff that uses exactly 2 cubes.

In questions 3 through 9, you may use no more than 1 N cube. Write each wff in wff1 – wff2 format.

3. Write a wff that uses exactly 3 cubes.

4. Write a wff that uses exactly 4 cubes.

5. Write a wff that uses exactly 5 cubes.

6. Write a wff that uses exactly 6 cubes.

7. Write a wff that uses exactly 7 cubes.

8. Write a wff that uses exactly 8 cubes.

9. Write a wff that uses exactly 9 cubes.

10. Write 2 different wffs that use all the cubes p p p K K

11. Write 5 different wffs that use all the cubes p p p p K K K

Wff Worksheet 3

Some of the expressions below are wffs. Rewrite each in "wff1 – wwf2" format. If the expression is not a wff, write "not a wff" or "X" or whatever.

Kp	______	AANqsr	______
ApNr	______	AKNrr	______
ArKpr	______	AErsCss	______
KNpq	______	ENrKsp	______
KApqErs	______	ECpqs	______
KCpqrs	______	EqKs	______
KssErs	______	EKrsq	______
KKpsr	______	CKqpr	______
CKKpsrr	______	CKrsNq	______
KEpNqr	______	KCKpqrqq	______
EKpqr	______	KqCqAqp	______
EKrsNrr	______	CqAqp	______
ACpqErs	______	KKrsApq	______
ArKCpqr	______	EKpqs	______
AKCpqrr	______	KNqCpr	______
AKprCqqr	______	ANKprq	______
Ekpq	______	KCprNq	______
KArsp	______	ANAqpr	______
CKArspq	______	CKpqArq	______
CqKArsp	______	ANqNq	______

What is a proof?

A proof is a sequence of steps that leads from one or more suppositions (each of which must be a wff) to a goal (which must also be a wff). Each step of the proof transforms wffs into other wffs, until the final wff, which is the goal, is reached. But there are specific rules that must be used to transform wffs. We will start with these rules shortly, along with sample proofs. We will use a special notation for these rules that takes a bit of getting used to, but that is really not all that complicated.

A proof begins with one or more *premises.* These are also called *suppositions* or *axioms* or *givens.* Mathematicians call them *antecedents.* The premises are on the left side of the arrow. The goal is on the right side of the arrow. All premises must be wffs. The goal must be a wff.

The body of a proof consists of a series of steps, shown with two columns. Each step must be numbered. The left column is always a wff. The right column is the justification for the wff. We always show the suppositions first, with the letter "s" (for "suppose") as the justification. (Don't worry if this isn't making much sense now – we are going to see a lot of proofs very soon, and they will all conform to these rules). The very last step of the proof must be the goal.

And then the final part of the proof is all suppositions, followed by a "/", and then a summary of all the rules used. Rules are not repeated.

So let's begin with our first rule, K out.

K out

We have a notation that we use to describe the rules used in WFF 'N PROOF. So let's jump right in:

```
Ko:   K-wff1-wff2  → wff1
      K-wff1-wff2  → wff2
```

We are going to see many many expressions with "→" . Here is how to interpret the first such expression above: "if we suppose Kwff1wff2, then we can assert wff1". Likewise, the 2nd expression is interpreted as "if we suppose Kwff1wff2, then we can assert wff2". This means that each K-wff consists of the K connective followed by exactly 2 wffs (recall rule three when forming a wff). The K out rule allows you to take out either the two wffs or even both of them if you want (but only one at a time).

And let's use this rule in a proof:

```
KpKrs -> p
-----------------------------------
1 | K-p-Krs    s
--+
2 | p          Ko, 1

KpKrs / Ko
```

Proof Requirements

This proof demonstrates the following requirements, which are required for every proof:

1. The expression above the vertical line, in this example `KpKrs -> p`, is what we are trying to prove. This is called a *theorem*, which is a mathematical term for something that can be proven.
2. A proof may have zero or more premises. (Premises are to the left of the theorem arrow, and must be wffs.) The goal must also be a wff.
3. Draw a vertical and a horizontal line.
4. Number each step of the proof.
5. Each proof step has 2 columns. The first column is a wff, and the 2nd column is the justification for that wff.
6. List all premises first, each on a separate line.
7. The justification for a premise is just the letter "s" (which stands for "suppose").
8. The justification for all steps after the premises must be a WFF 'N PROOF rule.
9. There must be one and only one justification for each step.
10. When specifying a rule, also specify the earlier proof step number(s) to which the rule applies. In this example, Ko is being applied to the wff in step one.
11. The wff in the last step of the proof must be the goal.
12. **Do not put dashes ("-") within your wffs. We do so in this book for clarity. In a tournament, though, your proof may be disqualified if you use dashes!**

The final line, `KpKrs / Ko`, is called the *solution*. All cubes used for all premises are listed to the left of the "/" separated by a comma, and all rules used in the proof are listed to the right separated by a comma. If a rule is used more than once in the proof, it is only listed once in the solution.

For now, write all wffs in your proof in wff1-wff2 format. Once you are quickly able to identify wff1 and wff2 you will no longer need to do this.

K out Examples

```
KKpqr  ->  p
-------------------------
1 | K-Kpq-r       s
--+
2 | K-p-q         Ko, 1
3 | p             Ko, 2

      KKpqr / Ko
```

```
KsKKpqr  ->  r
-------------------------
1 | K-s-KKpqr       s
--+
2 | K-Kpq-r         Ko, 1
3 | r               Ko, 2

      KsKKpqr / Ko
```

```
KNNqq  ->  NNq
------------------------
1 | K-NNq-q       s
--+
2 | NNq           Ko, 1

      KNNqq / Ko
```

```
KErqKrr  ->  Krq
-------------------------
1 | K-Erq-Krr       s
--+
2 | E-r-q           Ko, 1

     KErqKrr / Ko
```

K out Problems

Fill in the blanks:

```
KsCpq  ->  s
-------------------------
1 | ________      s
--+
2 | s             Ko, __

      KsCpq / Ko
```

```
KKpNrEpq  ->  Nr
-------------------------
1 | K-KpNr-Epq     s
--+
2 | _________      __, 1
3 | Nr             Ko, __

      _________ / Ko
```

```
KKKppqr  ->  p
-------------------------
1 | K-KKppq-r       s
--+
2 | _______         Ko, 1
3 | _______         Ko, 2
4 | p               __, 3

      KKKppqr / Ko
```

```
KqNKqq  ->  NKqq
-------------------------
1 | ______         s
--+
2 | ______         Ko, 1

      KqNKqq / Ko
```

K out Problems

Complete the following proofs:

`Krs  ->  s`

`KpNq  ->  Nq`

`KrKsq  ->  r`

`KKsqr  ->  r`

K out Problems

`KsKNpq -> s`

`KsKpq -> p`

`KKpqs -> p`

`KKNsrKpq -> Ns`

K out Problems

```
KKKrrsp  ->  p
```

```
KpKKrrs  ->  p
```

```
KpKKrrs  ->  s
```

```
KpKKrrs  ->  r
```

K out Problems

```
KKpKrsq  ->  r
```

```
KKpKrsq  ->  q
```

```
KKpKrsq  ->  Krs
```

```
KNNpKsq  ->  s
```

K in

The rule K in, abbreviated Ki, is the opposite of Ko: it combines 2 wffs into a single K wff.

```
Ki:  wff1, wwf2 → K-wff1-wff2
     wff1, wff2 → K-wff2-wff1
```

Order does not matter when creating the K-wff. To form a K-wff you must have each of the two wffs that are needed for that wff. Here are some examples:

```
p, q  ->  Kpq
-------------------------
1 | p              s
2 | q              s
--+
3 | K-p-q          Ki, 1, 2

      p,q / Ki
```

```
r, s  ->  KrKsr
-------------------------
1 | r              s
2 | s              s
--+
3 | K-s-r          Ki, 2, 1
4 | K-r-Ksr        Ki, 1, 3

      r,s / Ki
```

```
p, q  ->  KKpqKqp
-------------------------
1 | p              s
2 | q              s
--+
3 | K-p-q          Ki, 1, 2
4 | K-q-p          Ki, 2, 1
5 | K-Kpq-Kqp      Ki, 3, 4

      p,q / Ki
```

```
r, s, p, q  ->  KKKrspq
-------------------------
1 | r              s
2 | s              s
3 | p              s
4 | q              s
--+
5 | K-r-s          Ki, 1, 2
6 | K-Krs-p        Ki, 5, 3
7 | K-KKrsp-q      Ki, 6, 4

      r,s,p,q / Ki
```

K in

More than one kind of rule can be used within a proof!

```
Kpq, r  ->  Kqr
-------------------------
1 | K-p-q          s
2 | r              s
--+
3 | q              Ko, 1
4 | K-q-r          Ki, 3, 2

      Kpq,r / Ki,Ko
```

```
KNps, Kqr  ->  KrNp
-------------------------
1 | K-Np-s         s
2 | K-q-r          s
--+
3 | r              Ko, 2
4 | Np             Ko, 1
5 | K-r-Np         Ki, 3, 4

      KNps,Kqr / Ki,Ko
```

Fill in the blanks:

```
r, KKpqs  ->  Krp
-------------------------
1 | _______        s
2 | K-Kpq-s        s
--+
3 | K-p-q          Ko, 2
4 | _______        Ko, 3
5 | K-r-p          __, __, __
```

```
KKNqrs  ->  KNqs
-------------------------
1 | _______        s
--+
2 | K-Nq-r         Ko, 1
3 | s              __, 1
4 | _______        Ko, 2
5 | _______        Ki, 4, __
```

K in Problems

```
r, s  ->  Krs
```

```
p, q, r  ->  KKpqr
```

```
r, q  ->  KrKqr
```

```
Erq, p  ->  KpErq
```

```
Np, p  ->  KKpNpKNpp
```

```
r, p  ->  KKrpKpr
```

```
Crs, Kpq  ->  KKpqCrs
```

```
NNpq, s  ->  KNNps
```

K in Problems

You may now use other rules as well (i.e., Ko)

```
Kpq, r  ->  Krq                          Krs, KNps  ->  KNpr
```

```
KKpsr, Epq  ->  KrEpq                    r, s, Kpq  ->  KKprs
```

```
KKpqr, Nq  ->  KNqKpr
```

```
KNsq, KKrsp  ->  KrKrNs
```

```
KAprNs, q  ->  KqApr
```

```
KsKKpqr  ->  KKspr
```

Rp

Rp is the Repeat rule. This rule allows you to use the same wff twice within a proof.

```
Rp: wff1 → wff1
```

Here is an example of an illegal proof that shows when Rp is needed:

```
r  ->  Krr
-------------------------
1 | r             s
--+
2 | Krr           Ki, 1, 1      NO! Same prev. step may not be used twice!

      r / Ki
```

This proof is illegal because the Ki rule from proof step number 2 refers to the same earlier step number (that is, step number 1). This proof must instead use Rp:

```
r  ->  Krr
-------------------------
1 | r             s
--+
2 | r             Rp
2 | Krr           Ki, 1, 2

      r / Ki,Rp
```

Rp

Note that sometimes there are other ways in which you can reproduce a proof step other than using Rp:

```
Kpr  ->  Kpp
-------------------------
1 | Kpr            s
--+
2 | p              Ko, 1
3 | p              Ko, 1
4 | Kpp            Ki, 2, 3

      Kpr / Ki,Ko
```

That proof is perfectly valid, as is this one:

```
Kpr  ->  Kpp
-------------------------
1 | Kpr            s
--+
2 | p              Ko, 1
3 | p              Rp, 1
4 | Kpp            Ki, 2, 3

      Kpr / Rp,Ki,Ko
```

Rp Problems

Complete the following proofs:

`Np → KNpNpNp`

`r → KrKrr`

Complete the following proof twice, once with Rp, and the other without:

`Ksr → Kss`

Rp Problems

Complete the following proofs twice, once with Rp, and the other without:

`KpKsq → KsKss`

`KKrsq → KKrrKrr`

R cube and Required cubes

The R cube can also be used as a "wild" cube. This means that it can be substituted for any single rule in a proof. Here is an example of how to show that an R cube is used as a wild cube:

```
KpKrs -> p
-----------------------------------
1 | K-p-Krs    s
--+
2 | p          Ko, 1

KpKrs / R(Ko)
```

That is, show the rule that the R cube is replacing in parentheses. The R cube may only replace rules, and therefore it may never appear on the left side of the "/" within a solution.

Question: given that the R cube can also replace the rule "Rp", (as in R(Rp)), when would you want to *not* substitute R for Rp?

Required cubes

Within your solution, show all cubes in required by placing an arrow underneath. Suppose 3 K cubes and one o cube are in required. Your proof would look like this:

```
KpKrs -> p
-----------------------------------
1 | K-p-Krs    s
--+
2 | p          Ko, 1

KpKrs / Ko
↑ ↑     ↑↑
```

Showing required cubes is important when we discuss nonessentiality, later.

A in

A in, abbreviated Ai, is known as “buy one get one free”. With this rule you can create an A wff using just one other wff:

```
Ai: wff1 → A-wff1-wff2
           A-wff2-wff1
```

Here are some example proofs:

```
p  ->  Apq
-------------------------
1 | p              s
--+
2 | A-p-q          Ai, 1

      p / Ai
```

```
r  ->  AKKssqr
-------------------------
1 | r              s
--+
2 | A-KKssq-r      Ai, 1

      r / Ai
```

Notice how we can create and A wff using just one wff (as opposed to K in, which requires 2 wffs). When creating an A wff within a proof, find the “easier” wff, and use that along with A in to create your A wff.

A in Problems

Complete the following proofs:

```
Krs  ->  AqKrs
```

```
p  ->  ANpp
```

```
Ars  ->  ANqArs
```

```
Krs  ->  AsKsr
```

```
Kpq, r  ->  AKqrNp
```

```
Kpq, r  ->  KArpq
```

```
Krs  ->  AAAqqqr
```

```
KKpqr  ->  AKqpr
```

`Krs -> AsNKpNp`

`p, q -> ANNqKqp`

`p -> KAprAsp`

`Krs -> KArpAps`

```
KKpqr  ->  AKqrNs
```

```
r  ->  AKrrNNr
```

```
s  ->  AAsNpKrq
```

```
KNqr  ->  AKrNqNr
```

C out

The C out rule (abbreviated Co) says that if you have a C wff, and wff1 of that C wff, then you can assert wff2.

```
Co: C-wff1-wff2, wff1 → wff2
```

Here are some example proofs:

```
Kpq, p  ->  q
-------------------------
1 | K-p-q          s
2 | p              s
--+
3 | q              Co, 1, 2

      Kpq,p / Co
```

```
CNrs, Nr  ->  s
-------------------------
1 | C-Nr-s         s
2 | Nr             s
--+
3 | s              Co, 1, 2

      CNrs,Nr / Co
```

```
Cpqr, p, q  ->  r
-------------------------
1 | C-Kpq-r        s
2 | p              s
3 | q              s
--+
4 | K-p-q          Ki, 2, 3
5 | s              Co, 1, 4

      CKpqr,p,q / Ki,Co
```

```
CApqq, p  ->  q
-------------------------
1 | C-Apq-q        s
2 | p              s
--+
3 | A-p-q          Ai, 2
4 | q              Co, 1, 3

      CApqq,p / Ai,Co
```

C out Problems

Complete the following proofs:

```
Crs, r  ->  s
```

```
Cps, Kpq  ->  s
```

```
CNsr, Ns  ->  r
```

```
CKssq, s  ->  q
```

C out Problems

`CApqr, q  ->  r`

`Crq, Krs  ->  q`

`CrKpq, r  ->  p`

`CrKpq, Krq  ->  p`

C out Problems

```
CAqrKrp, q  ->  p                         CrCrp, r  ->  p
```

```
CKrqs, r, q  ->  AsNr                      Crs, Krq  ->  KANrsANNps
```

C out Problems

```
CArrp, Cpq, r  ->  q
```

```
Krs, CrCsq  ->  q
```

```
CKrsq, r, s  ->  ANNNqq
```

```
CAKKppqrCrs, r  ->  Asq
```

E out

The E out rule (Eo) converts an E wff into a C wff:

```
Eo: E-wff1-wff2 → C-wff1-wff2
    E-wff1-wff2 → C-wff2-wff1
```

Here are some example proofs:

```
Epq, q  ->  p
-------------------------
1 | E-p-q           s
2 | q               s
--+
3 | C-q-p           Eo, 1
4 | p               Co, 3, 2

      Epq,q / Eo,Co
```

```
Ers, r  ->  s
-------------------------
1 | E-r-s           s
2 | r               s
--+
3 | C-r-s           Eo, 1
4 | s               Co, 3, 2

      Ers,r / Eo,Co
```

```
CrEpr, r  ->  p
-------------------------
1 | C-r-Epr         s
2 | r               s
--+
3 | E-p-r           Co, 1, 2
4 | C-r-p           Eo, 3
5 | p               Co, 4, 2

      CrEpr,r / Eo,Co
```

```
ErApq, q  ->  r
-------------------------
1 | E-r-Apq         s
2 | q               s
--+
3 | A-p-q           Ai, 2
4 | C-Apq-r         Eo, 1
5 | r               Co, 4, 3

    ErApq,q / Ai,Eo,Co
```

E out Problems

Complete the following proofs:

```
EKssq, s  ->  q
```

```
Eqr, Krs  ->  q
```

```
ErKpq, r  ->  p
```

```
EKpqr, Krq  ->  p
```

E out Problems

```
EKrpAqr, q  ->  p
```

```
ErCrp, r  ->  p
```

```
ErEpr, r  ->  p
```

```
Krs, ErCsq  ->  q
```

E in

The E in rule (Ei) combines 2 C wffs into an E wff:

```
Ei: C-wff1-wff2, C-wff2-wff1 → E-wff1-wff2
    C-wff1-wff2, C-wff2-wff1 → E-wff2-wff1
```

Here are some example proofs:

```
Cqp, Cpq  ->  Epq
-------------------------
1 | C-q-p         s
2 | C-p-q         s
--+
3 | E-p-q         Ei, 1, 2

      Cqp,Cpq / Ei
```

```
CKrsq, CqKrs  ->  EKrsq
-------------------------
1 | C-Krs-q       s
2 | C-q-Krs       s
--+
3 | E-Krs-q       Ei, 1, 2

      CKrsq,CqKrs / Ei
```

E in Problems

Complete the following proofs:

`KCpqCqp -> Eqp`

`ErCpq, r, Cqp -> Epq`

`CrCsr, CpCrs, Kpr -> Ers`

`KErCpqErCqp, r -> Eqp`

Nonessentiality

There is a rule in WFF 'N PROOF that says

All cubes in <u>required</u> must be used *essentially* within a proof.

What does this mean? Suppose an opponent presents the following proof and solution to you, <u>where all the cubes in the solution are in required:</u>

```
p, q, r  ->  Kpq
-------------------------
1 | p              s
2 | q              s
3 | r              s
--+
4 | Kpq            Ki, 1, 2

      p,q,r / Ki
      ↑ ↑ ↑   ↑↑
```

Looking at this proof, you realize that the r cube is not needed, and that the proof could be rewritten as

```
p, q  ->  Kpq
-------------------------
1 | p              s
2 | q              s
--+
3 | Kpq            Ki, 1, 2

      p,q / Ki
```

Thus we say that the r cube is *nonessential*. We have shown that a premise is nonessential.

Nonessentiality

Now consider another example, where once again <u>all the cubes in the solution are in required</u>:

```
p, q  ->  Kpq
-------------------------
1 | p              s
2 | q              s
--+
3 | ApErs          Ai, 1
4 | Kpq            Ki, 1, 2

      p,q / Ai, Ki
      ↑ ↑   ↑↑  ↑↑
```

Looking at this proof, you realize that step number 3 is not needed, and thus the Ai rule, and therefore the A cube, is not needed in the solution. The proof can be rewritten as

```
p, q  ->  Kpq
-------------------------
1 | p              s
2 | q              s
--+
3 | Kpq            Ki, 1, 2

      p,q / Ki
```

Rules for writing a proof of nonessentiality

You can disqualify a proof if you can show that one or more cubes in required are *nonessential* in that proof. To do so, you must rewrite the proof without one or more required cubes. (We call that a "proof of nonessentiality".) But there are some restrictions:

1. You may not change or add any premises. You can completely ignore a premise, but you cannot add or change a premise.

2. You cannot introduce any more cubes into the proof. For example, you may not introduce cubes from permitted into your proof of non-essentiality.

3. If the wildcard cube "R" is used in the solution, you cannot change the rule to which it refers.

Nonessentiality

Remember, only a rule or a premise can be proven nonessential. Rewriting a proof just to eliminate a proof step may not be enough!

Why should we always indicate required cubes in a solution?

Because if you don't, your opponent will!

Consider the following proof, where 2 K cubes are in required:

```
Krs, Kpq  ->  r
------------------------
1 | Krs            s
2 | Kpq            s
--+
3 | r              Ko, 1

      Krs,Kpq / Ko
```

Note that no required cubes have been indicated in the solution. Because of this, the following proof of nonessentiality can be presented. (We assume both required cubes are premises.)

```
Krs ->  r
------------------------
1 | Krs            s
--+
2 | r              Ko, 1

      Krs / Ko
      ↑     ↑
```

However, had the original proof and solution been presented as follows, no nonessentiality proof could be written:

```
Krs, Kpq  ->  r
------------------------
1 | Krs            s
2 | Kpq            s
--+
3 | r              Ko, 1

    Krs,Kpq / Ko
    ↑         ↑
```

Nonessentiality Problems

Write a proof of nonessentiality to the right of each proof below.

```
KKpsq, Cpq  ->  q
-------------------------
1 | KKpsq          s
2 | Cpq            s
--+
3 | Kps            Ko, 1
4 | p              Ko, 4
5 | q              Co, 2, 4

      KKpsq,Cpq / Co,Ko
                  ↑  ↑
```

```
r, CArsq  ->  Aqr
-------------------------
1 | r              s
2 | CArsq          s
--+
3 | Ars            Ai, 1
4 | q              Co, 2, 3
5 | Aqr            Ai, 4

      r,CArsq / Ai,Co
                ↑  ↑
```