An example in NQTHM: Ramsey's Theorem

Matt Kaufmann *

Computational Logic, Inc.
1717 W. 6th Street, Suite 290
Austin, TX 78703
kaufmann@cli.com

September 15, 1992

Abstract

We present here a proof of Ramsey's Theorem for exponent 2 using the Boyer-Moore theorem prover. The presentation is intended to be in a style to assist those who want to learn how to improve their effectiveness in using the Boyer-Moore logic and theorem prover.

*sponsored in part by the Defense Advanced Research Projects Agency, DARPA Orders 6082 and 9151. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., the Defense Advanced Research Projects Agency or the U.S. Government. Earlier related work was supported by ONR Contract N00014-81-K-0634.

1 Introduction

Beginning users of the Boyer-Moore theorem prover often have some difficulty in getting a feel for how to use this system. In this note I show how I used the Boyer-Moore logic and theorem prover to formalize and prove a particular theorem. That is, I chronicle here the development of a mechanically-checked proof of the finite version of Ramsey's Theorem for exponent 2, as contained in [3][1].

In the next section I'll give an introduction to a formalization of this theorem in the Boyer-Moore logic, and in the third and final section I'll give a carefully annotated list of Boyer-Moore events comprising a statement and proof of Ramsey's theorem. The remainder of this introduction presents a statement and proof of (this version of) Ramsey's Theorem in ordinary mathematical notation.

The reader who wishes simply to see the events ultimately submitted to the Boyer-Moore theorem prover will find that these are exactly all of the lower-case displays in `typewriter font` in the final section. Finally, I thank my colleagues here at Computational Logic Inc. for making this a great place to work.

1.1 A statement of Ramsey's Theorem

Suppose one has an undirected graph G. A subset H of this graph is a *clique* if every pair of two (distinct) nodes from the set H is joined by an edge in G. A subset H of this graph is *independent* if no pair from H is joined by an edge in G. Ramsey's Theorem then states that for every ordered pair $< p, q >$ of natural numbers there is a natural number N such that for every undirected graph G possessing N nodes, there is either a clique of G having p nodes or an independent subset of G having q nodes.

It is convenient to introduce one more notion at this point. A subset of a graph is *homogeneous* iff it is either a clique or an independent set. We will use this notion in our formalization. For now let's just say that if one takes $p = q$ in the statement above, one gets an equivalent formulation that is a common one for Ramsey's Theorem: For any natural number p there is a natural number N (depending only on p) with the following property: every (undirected) graph with at least N nodes has a homogeneous subset of size p.

1.2 A proof of Ramsey's Theorem

A standard proof of this theorem is by strong induction on $p + q$ and goes as follows. If p or q equals 0 then the conclusion is obvious, as we may take the empty set for our clique or independent set (respectively). So suppose $p, q > 0$. Choose $N_1 > 0$ and $N_2 > 0$ respectively to satisfy the conclusion of the theorem for $(p - 1) + q$ and $p + (q - 1)$. We claim that the sum N of N_1 and N_2 is the desired N. For suppose that G has N nodes, let a be an arbitrary node

[1] I thank Jim Schmerl for bringing this proof to my attention.

in G, and partition the rest of G into the sets S_1 and S_2 of nodes which are respectively connected / not connected to a by an edge. Then either S_1 has at least N_1 elements or S_2 has at least N_2 elements. The proofs of these two cases are symmetric, so let us concentrate on the first. Suppose then that S_1 has at least N_1 elements. By the inductive hypothesis, either there is a subset C of S_1 of size $p-1$ which is a clique of G or else there is a subset I of S_1 which is independent in G. Then either $C \cup \{a\}$ is a clique of size p or I is an independent set of size q, as desired.

2 Introduction to a formalization of Ramsey's Theorem in the Boyer-Moore logic

In this section I'll give an outline of a formalization of Ramsey's Theorem in the Boyer-Moore logic. The first subsection below is an informal description of the "types" of the variables occurring in the ultimate statement of the theorem and in auxiliary definitions. The second subsection describes the supporting functions, while the third is a formalization of Ramsey's theorem which uses these functions.

2.1 The "data types"

The words "data types" are quoted above because in fact there are no real types in the Boyer-Moore logic. So to be more precise, I should say how the reader may wish to think of the variables in the various definitions below, in order to understand the functions described below. In this formulation we represent sets and relations using lists. Thus, `DOMAIN` is a list which we think of as being a one-one enumeration of a finite set, and `PAIRS` is a list of pairs which we think of as representing a finite binary relation.[2] If `DOMAIN` has no repetitions, then its `LENGTH` is just the cardinality of the set which it represents (where non-lists are viewed as representing the empty set). Finally, the variables `p` and `q` are intended to represent natural numbers; non-numeric values of `p` and `q` should be thought of as representing the number `0`.

2.2 The functions

Now I'll briefly and informally describe the functions that will be used to state the formalization of Ramsey's Theorem to come. (Careful definitions will be presented in the next subsection.) The first thing to notice is that there are existential quantifiers in the statement of Ramsey's Theorem and yet the Boyer-Moore logic does not admit existential quantification. The standard solution to

[2]Formally, a predicate `RELATED` is defined which holds of a pair `(CONS X Y)` and a list exactly when either `(CONS X Y)` or `(CONS Y X)` (or both) belongs to the list.

such dilemmas is to define functions which provide the values required by the existential quantifiers.[3] For example, I'll define a 2-place function RAMSEY such that for all natural numbers p and q, RAMSEY(p, q) equals a number N which satisfies the conclusion of the theorem. I'll also define a function WIT which can be thought of as a "WITnessing" function which picks out the desired clique or independent subset of the given list DOMAIN, with respect to the graph on this set represented by the list PAIRS of (presumably) ordered pairs. More precisely, WIT returns an ordered pair whose first component (CAR) is this set and whose second component (CDR) is either 1 or 2 according to whether the set is a clique or an independent set (respectively). The function GOOD-HOM-SET asserts correctness of the witnessing function WIT on its arguments, in that this function produces a clique or independent set which is as large as required, depending on whether the argument FLG is 1 or 2, respectively. The following Lisp-style syntax is used for presenting definitions in the Boyer-Moore logic to the theorem prover. It defines GOOD-HOM-SET to be a 5-place function which, for arguments named PAIRS, DOMAIN, P, Q, and FLG, which returns the boolean conjunction of two values. The first is the value of the function HOMOGENEOUS (described below) on the three arguments shown below, namely: a set returned by WIT together with PAIRS and FLG (where we think of FLG as being 1 or 2, as described above). The second says that this WIT set is at least as large as p or q (according to whether FLG suggests that we are looking for a clique or an independent set).

```
(defn good-hom-set (pairs domain p q flg)
  ;; flg is 1 or 2
  (and (homogeneous (car (wit pairs domain p q))
                    pairs
                    flg)
       (not (lessp (length (car (wit pairs domain p q)))
                   (if (equal flg 1) p q)))))
```

Here (HOMOGENEOUS S PAIRS 1) is true, i.e. equals T, if the list S is a clique for pairs, i.e. for each pair from S, either it or its reverse belongs to PAIRS. Similarly, (HOMOGENEOUS S PAIRS 2) is true if S is an independent set with respect to the relation represented by PAIRS. RAMSEY is a function which returns an integer which (according to the theorem above) is "large enough"; we called it "N" in the subsection before last. (In fact (RAMSEY X Y) is the binomial coefficient $P + Q$ *choose* P, though this isn't proved here.) The function LEQ is just "less than or equal", at least on the natural numbers. The function SETP holds of a list if and only if the list contains no duplicates.

[3] Some would call these functions *Skolem functions.*

2.3 The formalized theorem

Finally, let us state the theorem. It says that if `DOMAIN` is sufficiently large, then `(CAR (WIT PAIRS DOMAIN P Q))` is (1) a subset of `DOMAIN` which is (2) a clique or independent set of sufficient size and (3) represents a set (assuming that `DOMAIN` represents a set).[4]

```
THEOREM.  RAMSEY-THEOREM-2
(IMPLIES (LEQ (RAMSEY P Q)
              (LENGTH DOMAIN))
         (AND (SUBSETP (CAR (WIT PAIRS DOMAIN P Q))
                       DOMAIN)
              (GOOD-HOM-SET PAIRS DOMAIN P Q
                            (CDR (WIT PAIRS DOMAIN P Q)))
              (IMPLIES (SETP DOMAIN)
                       (SETP (CAR (WIT PAIRS DOMAIN P Q))))))
```

3 The proof

The current effort took about 7 hours, but that number is misleading since I did this once before [4] and used nearly the same definitions this time, as well as some of the same lemmas. (I won't refer to that other proof again after this paragraph, however.) Nevertheless, except for the subset lemmas I think it's fair to say that I rediscovered the necessary lemmas, and in fact the new proof is considerably cleaner than my earlier one. Moreover, the new proof replays in the unadorned Boyer-Moore theorem prover, while the old proof made extensive use of my "proof-checker" interactive enhancement[5] [5]. I did, however, use the proof-checker quite a lot to help me discover useful lemmas to prove in the current effort. Interestingly, the new proof replays significantly faster than the old one (when both are done on a Sun 3/60 with 16 megabytes main memory): the time triple [miscellaneous time, prove time, IO time], measured in seconds is [11.9 181.3 26.5] for a run of the new proof, as compared to [27.9 328.0 38.8] for the old version. Normally I expect proofs which use my proof-checker to replay more quickly than corresponding ones that do not, since the user has presumably

[4] Notice that `SUBSETP` is really a notion defined on lists rather than sets. In order to know that our representation faithfully represents Ramsey's theorem, we need to know that if the given list represents a set in the sense that it has no duplicates, then so does the purported homogeneous set (list).

[5] That system allows one to create PROVE-LEMMA events by first submitting the proposed theorem and then interactively giving various proof commands. These commands range from "low-level" commands which invoke a particular definition or rewrite rule to "medium-level" commands invoking simplification, to "high-level" commands which actually call the Boyer-Moore theorem prover.

programmed part of the proof in the former case and the instructions are actual input, while for the "fully automatic" proof the heuristic must start from scratch in the replay. I guess that's an indication of how much better my new proof is than the older one.

Here then is the sequence of events, with comments to explain some of what was going on when I did this proof. The first subsection below contains the definitions. The theorem breaks naturally into three pieces, and the second through fourth subsections present the proofs of the respective pieces, culminating in the theorem.

3.1 The definitions

One starts by initializing the theorem prover's database:

```
(boot-strap nqthm)
```

Recall that **RAMSEY**(p, q) is supposed to pick out a "sufficiently large" integer N such that every graph of size N has a clique of size p or an independent set of size q. The following recursive definition mirrors the proof given in Section 1. This can be explained a bit more as follows. Let's say that a graph G has the *p, q – homogeneity property* if it has either a clique of size p or an independent set of size q. Then the inductive step of the proof applies the inductive hypothesis to a subgraph with either the *p-1, q – homogeneity property* or the *p, q-1 – homogeneity property*. The partitioning that goes on shows that we need a graph of size at least N_1+N_2, where N_1 and N_2 are the numbers given by the inductive hypothesis which guarantee that the respective homogeneity properties above hold for graphs with these respective numbers of nodes. In other words, we want **RAMSEY**(p, q) to be at least as large as the sum of **RAMSEY**$(p-1, q)$ and **RAMSEY**$(p, q-1)$. And that's just what the recursive definition below does for us!

```
(defn ramsey (p q)
  (if (zerop p)
      1
      (if (zerop q)
          1
          (plus (ramsey (sub1 p) q)
                (ramsey p (sub1 q)))))
  ;; A hint to the prover (see below)
  ((lessp (plus p q))))
```

The definition above has a hint to help the theorem prover "prove" that this function is total. That is, the system is being asked to check that the term `(PLUS P Q)` decreases in the sense of `LESSP` on each recursive call of the function `RAMSEY`.

The following boolean-valued function says that `I` and `J` are connected by an edge in the graph represented by `PAIRS`. Since we are interested in undirected graphs, we view `PAIRS` as representing the relation that is its symmetric closure.

```
(defn related (i j pairs)
  (or (member (cons i j) pairs)
      (member (cons j i) pairs)))
```

Informally speaking, the next function takes a node `N` and a list `REST` and returns an ordered pair of lists whose union is `REST` such that each node in the first component of the result is connected to `N` and each node in the second component is unconnected to `N`.

```
(defn partition (n rest pairs)
  (if (listp rest)
      (if (related n (car rest) pairs)
          (cons (cons (car rest)
                      (car (partition n (cdr rest) pairs)))
                (cdr (partition n (cdr rest) pairs)))
          (cons (car (partition n (cdr rest) pairs))
                (cons (car rest)
                      (cdr (partition n (cdr rest) pairs)))))
      (cons nil nil)))
```

Here is the usual length function.

```
(defn length (lst)
  (if (listp lst)
      (add1 (length (cdr lst)))
      0))
```

The following function constructs the desired clique or independent set for the set represented by `DOMAIN` with graph structure given by the binary relation represented by `PAIRS`. The following definition uses a useful abbreviation `LET` which is just like the one in pure Lisp: for example, `(LET ((X 2) (Y 3)) (PLUS X Y Z))` abbreviates the term (PLUS 2 3 Z).[6] The structure of this definition mirrors the proof given in the first subsection of this note.

```
(defn wit (pairs domain p q)
  ;; returns (cons set flag) where flag is 1 or 2
  (if (listp domain)
      (if (zerop p)
          (cons nil 1)
        (if (zerop q)
            (cons nil 2)
          (let ((set1 (car (partition (car domain)
                                      (cdr domain)
                                      pairs)))
                (set2 (cdr (partition (car domain)
                                      (cdr domain)
                                      pairs))))
            (if (lessp (length set1)
                       (ramsey (sub1 p) q))
                ;; then use set2 to form a clique or
                ;;  an independent set
                (let ((wit-set2 (wit pairs set2 p (sub1 q))))
                  (if (equal (cdr wit-set2) 1)
                      wit-set2
                    (cons (cons (car domain)
                                (car wit-set2))
                          2)))
              ;; otherwise use set1 to form an independent set
              ;;  or a clique
              (let ((wit-set1 (wit pairs set1 (sub1 p) q)))
                (if (equal (cdr wit-set1) 2)
                    wit-set1
                  (cons (cons (car domain)
                              (car wit-set1))
                        1)))))))
    (cons nil 1))
  ((lessp (plus p q))))
```

I should note that at first I omitted the (LISTP DOMAIN) test at the top of this definition. However, that caused the first main lemma, which says that (CAR (WIT PAIRS DOMAIN P Q)) is a subset of DOMAIN, to be false in general.

[6] In fact this abbreviation is implemented in the proof-checker, but I eliminated it before submitting the definition to the unadorned Boyer-Moore theorem prover.

In fact that lemma becomes true if one adds a hypothesis which says that (RAMSEY P Q) is less than or equal to (LENGTH DOMAIN), and I did that proof. But it was inconvenient to restrict this lemma so, since I used it later in the proof to prove other lemmas. What's more, the omission of the (LISTP DOMAIN) test caused failure of the lemma CDR-WIT-IS-1-OR-2 (see the paragraph above that lemma on page 19). What's more, that "buggy" version of the definition of WIT was somewhat unreasonable anyhow, since it's a bit strange (though legal) to be taking the CAR of DOMAIN when DOMAIN is not known to be a LISTP (that is, not known to be a cons pair). I should mention that in proving the version of the subset lemma just mentioned, I proved lemmas LENGTH-PARTITION and RAMSEY-NOT-ZERO that were needed anyhow for later parts of the proof; I'll mention this again when we get there.

The following definitions are needed for our statement of Ramsey's Theorem. The function HOMOGENEOUS1 is a predicate which says whether every element of DOMAIN shares an edge with node N in the graph represented by PAIRS, at least when the argument FLG is 1; otherwise it says that every element of DOMAIN shares *no* edge with N. The function HOMOGENEOUS whose definition follows that is a predicate saying that every element of DOMAIN shares an edge with every other (or with no other, again depending on FLG). SUBSETP has the obvious meaning.

```
(defn homogeneous1 (n domain pairs flg)
  (if (listp domain)
      (and (if (equal flg 1)
               (related n (car domain) pairs)
               (not (related n (car domain) pairs)))
           (homogeneous1 n (cdr domain) pairs flg))
      t))

(defn homogeneous (domain pairs flg)
  (if (listp domain)
      (and (homogeneous1 (car domain) (cdr domain) pairs flg)
           (homogeneous (cdr domain) pairs flg))
      t))

(defn subsetp (x y)
  (if (nlistp x)
      t
      (if (member (car x) y)
          (subsetp (cdr x) y)
          f)))
```

The following two definitions were actually presented to the system later in our proof development. Hence we'll state them again below where they actually occur. (Each line in the following two definitions starts with a semicolon to emphasize that we did not submit the definitions at this point; similarly for the statement of the main theorem, which immediately follows these remaining two definitions.) The first defines a good homogeneous set to be one that's "big enough". We expect `FLG` to be `1` or `2` according to the same convention discussed above.[7]

```
;(DEFN GOOD-HOM-SET (PAIRS DOMAIN P Q FLG)
;  (AND (HOMOGENEOUS (CAR (WIT PAIRS DOMAIN P Q))
;                    PAIRS
;                    FLG)
;       (NOT (LESSP (LENGTH (CAR (WIT PAIRS DOMAIN P Q)))
;                   (IF (EQUAL FLG 1) P Q)))))
```

The remaining definition simply says that a given list has no duplicates.

```
;(DEFN SETP (X)
;  (IF (LISTP X)
;      (AND (NOT (MEMBER (CAR X) (CDR X)))
;           (SETP (CDR X)))
;    T))
```

The proof of the formalization of Ramsey's Theorem, as presented at the end of the previous section, naturally splits into three cases corresponding to the three conjuncts of the conclusion. We now consider these in turn.

3.2 The first part of the proof: the witness set is a subset

Our goal now is to prove the first of the three parts of the theorem, namely `(SUBSETP (CAR (WIT PAIRS DOMAIN P Q)) DOMAIN)`. An attempt to prove this lemma, which I'll call SUBSETP-HOM-SET-DOMAIN, shows that one needs some facts about `SUBSETP`. So, I proved some obvious facts at this stage with the expectation that some would help. (I actually developed these previously, though I don't remember if they were in fact developed for the Ramsey proof in particular!)

[7] These definitions are in upper case for the sake of the convention promised in the third paragraph of Section 1.

```
(prove-lemma member-cons (rewrite)
  (implies (member a l)
           (member a (cons x l))))

(prove-lemma subsetp-cons (rewrite)
  (implies (subsetp l m)
           (subsetp l (cons a m))))

(prove-lemma subsetp-reflexivity (rewrite)
  (subsetp x x))

(prove-lemma cdr-subsetp (rewrite)
  (subsetp (cdr x) x))

(prove-lemma member-subsetp (rewrite)
  (implies (and (member x y)
                (subsetp y z))
           (member x z)))

(prove-lemma subsetp-is-transitive (rewrite)
  (implies (and (subsetp x y)
                (subsetp y z))
           (subsetp x z)))
```

Next I tried again to prove SUBSETP-HOM-SET-DOMAIN. The theorem prover did a generalization, which is usually bad news; it seemed to need a fact about `SUBSETP` and `PARTITION`, together with transitivity of `SUBSETP` (proved above). This fact about `SUBSETP` and `PARTITION` is SUBSETP-HOM-SET-DOMAIN-1 below, which follows from the next lemma, SUBSETP-CDR-PARTITION. In fact the version of SUBSETP-CDR-PARTITION I needed was with `(CONS X Z)` as the second argument to `SUBSETP`, but by the lemma SUBSETP-CONS above, I (correctly) expected that this more general version (without `CONS`) would suffice. More general versions of lemmas are often easier to prove, not to mention more applicable, and hence are often to be preferred. (So why didn't I state SUBSETP-HOM-SET-DOMAIN-1 with similar generality? I really don't know!!)

```
(prove-lemma subsetp-cdr-partition (rewrite)
  (subsetp (cdr (partition x z pairs))
           z))

(prove-lemma subsetp-hom-set-domain-1 (rewrite)
  (implies
```

```
    (subsetp
      (car (wit pairs (cdr (partition x z pairs)) p q))
      (cdr (partition x z pairs)))
    (subsetp
      (car (wit pairs (cdr (partition x z pairs)) p q))
      (cons x z)))
  ;; The USE hint below was needed because of the free variable
  ;; in the statement of transitivity of SUBSETP.  The DISABLE
  ;; hint was there so that the use of SUBSETP-CDR-PARTITION
  ;; wasn't rewritten to T.
  ((use (subsetp-cdr-partition))
   (disable subsetp-cdr-partition)))
```

Trying the proof of SUBSETP-HOM-SET-DOMAIN once again, I found that the goals pushed for induction all followed either from the analog of the above lemma for `(CAR (PARTITION ....))` [8] or from the fact that `RAMSEY` never returns `0`.

```
(prove-lemma subsetp-car-partition (rewrite)
  (subsetp (car (partition x z pairs))
           z))

(prove-lemma subsetp-hom-set-domain-2 (rewrite)
  (implies
    (subsetp
      (car (wit pairs (car (partition x z pairs)) p q))
      (car (partition x z pairs)))
    (subsetp
      (car (wit pairs (car (partition x z pairs)) p q))
      (cons x z)))
  ((use (subsetp-car-partition))
   (disable subsetp-car-partition)))
```

And now I was ready for the prover to finish off the first of the three main lemmas:

```
(prove-lemma subsetp-hom-set-domain (rewrite)
  (subsetp (car (wit pairs domain p q))
           domain))
```

[8] in fact I used an editor to "create" this analog

3.3 The second part of the proof: the witness set is large enough and homogeneous

The second of the three main goals was to show that under the "sufficient size" hypothesis, one gets a sufficiently large homogeneous set. (The upper case and semicolons emphasize that this event actually occurs later in the event list submitted to the prover; hence we'll see this event again later.)

```
;(PROVE-LEMMA WIT-YIELDS-GOOD-HOM-SET (REWRITE)
;  (IMPLIES (NOT (LESSP (LENGTH DOMAIN)
;                       (RAMSEY P Q)))
;           (GOOD-HOM-SET PAIRS DOMAIN P Q
;                         (CDR (WIT PAIRS DOMAIN P Q)))))
```

Recall that for convenience, a "good" homogeneous set was defined to be one that's "big enough":

```
(defn good-hom-set (pairs domain p q flg)
  (and (homogeneous (car (wit pairs domain p q))
                    pairs
                    flg)
       (not (lessp (length (car (wit pairs domain p q)))
                   (if (equal flg 1) p q)))))
```

I started the proof of this lemma by using the proof-checker (see footnote on page 5), as an aid to discovering the structure of the proof. I doubt that this was necessary in order for me to be able to complete the proof, but I did feel that it was helpful to me. Actually, I hoped to be able to develop nice rewrite rules so that I wouldn't have to get too involved with every little detail, perhaps so that the final proof would involve only standard Boyer-Moore events that did not involve the proof-checker enhancement. Hence I proceeded in the proof-checker without doing much hand-proving of small details. I did feel free to use the PROVE command to call the theorem prover, however, since an automatic proof would behave similarly at such "stages" anyhow.

Thus I started by immediately using (heuristic) induction; in fact I used a command called INDUCT which replaces the goal by heuristically generated base and induction steps. I then immediately used a command called PROVE, which submits the current goal to the Boyer-Moore prover, on three of the 7 subgoals. Those subgoals corresponded to the "base cases", i.e. when `P` or `Q` is `ZEROP` (i.e. 0 or not a natural number, which we think of as representing 0 anyhow) or when `DOMAIN` is not a `LISTP` (non-empty list). I then took the first of the four remaining cases, opened up `GOOD-HOM-SET` (with `IF`s propogating up, using the propositional simplification command S-PROP), and then used the command SPLIT to cause a propositional split into 8 subgoals. The first time I saw a goal that I couldn't pass up (on an earlier attempt that I didn't document so well, actually) was one whose conclusion was:

```
(HOMOGENEOUS1 (CAR DOMAIN)
              (CAR (WIT PAIRS
                        (CDR (PARTITION (CAR DOMAIN)
                                        (CDR DOMAIN)
                                        PAIRS))
                        P
                        (SUB1 Q)))
              PAIRS 2)
```

This suggested that I prove the following lemma. Notice that in this lemma below, the variable `DOMAIN` is *free*, i.e. appears in the hypothesis but not in the conclusion. That makes this a rather useless automatic rewrite rule in many cases, but it's convenient this way for later use by the REWRITE command in the proof-checker.

```
(prove-lemma homogeneous1-subset (rewrite)
  (implies (and (subsetp x domain)
                (homogeneous1 elt domain pairs flg))
           (homogeneous1 elt x pairs flg)))
```

Next I entered a fresh session of the proof-checker with the following generalized version of the capitalized goal above,

```
(*)   (HOMOGENEOUS1 ELT
                    (CAR (WIT PAIRS
                              (CDR (PARTITION ELT DOM PAIRS))
                              P
                              Q))
                    PAIRS 2)
```

I attempted to manually apply the rewrite rule HOMOGENEOUS1-SUBSET above to prove this goal. To be precise, I applied HOMOGENEOUS1-SUBSET with the free variable `DOMAIN` instantiated to `(CDR (PARTITION ELT DOM PAIRS))`. (A misguided attempt to instantiate it instead with `DOM` lead to the requirement `(HOMOGENEOUS1 ELT DOM PAIRS 2)`, which is not a theorem!) This use of the REWRITE command immediately demonstrated the need for two rewrite rules, corresponding to the hypotheses of the rewrite rule HOMOGENEOUS1-SUBSET that was applied. One of the two rules apparently needed was the lemma SUBSETP-HOM-SET-DOMAIN proved in the subsection above. Here is the other rule needed in order to finish the proof the goal (*) displayed above.[9]

```
(prove-lemma homogeneous1-cdr-partition (rewrite)
  (homogeneous1 elt (cdr (partition elt dom pairs)) pairs 2))
```

Returning to the interactive proof of our main goal[10], I saw that I needed a lemma relating the length of a list to the lengths of the pieces of a partition of the list. As I mentioned above, I already knew about this lemma from the first proof I did of SUBSETP-HOM-SET-DOMAIN under a previous definition of `WIT`. The first lemma below is in a form that's easy for the prover to prove, while the second lemma is more appropriate for the rewriter to apply but (I think) harder to prove directly, though it follows easily from the first one.

```
(prove-lemma length-partition-1 nil
  (equal (length z)
         (plus (length (car (partition x z pairs)))
               (length (cdr (partition x z pairs))))))

(prove-lemma length-partition (rewrite)
  (equal (length (car (partition x z pairs)))
         (difference (length z)
                     (length (cdr (partition x z pairs)))))
  ((use (length-partition-1))))
```

At this point I went back in to the interactive prover and invoked the PROVE command a couple of times and then (REPEAT PROVE)[11], which instructed the system to call the prover on each goal until a proof failed (or was interrupted). This finished the rest of the 8 subgoals of the first of the 4 remaining

[9]This paragraph is harder to read than is the corresponding application of the proof-checker. The proof-checker has a command SHOW-REWRITES which shows the rewrite rules that apply, in this case HOMOGENEOUS1-SUBSET, as well as the hypotheses to relieve (which will result in new subgoals generated). The free variables are prefixed by dollar signs by SHOW-REWRITES, and the user can provide the instantiations for these. One of the two created subgoals would immediately yield to the REWRITE command, using the rewrite rule SUBSETP-HOM-SET-DOMAIN proved in the preceding subsection. The other would immediately suggest the lemma HOMOGENEOUS1-CDR-PARTITION. See [5] for details.

[10]yes, one can resume suspended interactive proofs with the proof-checker

[11]REPEAT is an example of a user-defined *macro command* in the proof-checker, in the tradition of LCF [2] and NuPrl [1]

goals, and started the second of the four remaining main goals (while I was doing some of this typing in another buffer!). The proof of that goal was not succeeding (the prover went into induction, actually), because the prover didn't realize the following fact:

```
(EQUAL
 (PLUS (RAMSEY V (ADD1 W)) (RAMSEY (ADD1 V) W))
 0)
=
(FALSE) .
```

I decided that the best way to try to handle this was with the following linear lemma (yes, even though the types are `(REWRITE)`, this is a linear rule, not a replacement rule; see [6]):

```
(prove-lemma ramsey-not-zero (rewrite)
  ;; Used as a linear lemma
  (lessp 0 (ramsey p q)))
```

This was the other place referred to previously where I already knew about this lemma from the first proof I did of SUBSETP-HOM-SET-DOMAIN under a previous definition of `WIT`. At this point I felt bold and was ready to try the proof of WIT-YIELDS-GOOD-HOM-SET (the goal of this subsection), without the interactive system, that is. Recall though that I used the lemma HOMOGENEOUS1-SUBSET manually in the interactive proof described above, and it has a free variable which I told the system how to instantiate. So now I wanted to prove that goal, labeled (*) above, as a lemma. This would be a bit easier to do interactively since then I wouldn't need to provide instantiations for the non-free variables — the instructions hint would simply be

```
(INSTRUCTIONS (REWRITE HOMOGENEOUS1-SUBSET
                 (($DOMAIN
                     (CDR (PARTITION ELT DOM PAIRS)))))
              (REWRITE SUBSETP-HOM-SET-DOMAIN)
              (REWRITE HOMOGENEOUS1-CDR-PARTITION))
```

— but what the heck, I was shooting for a proof that would replay in the Boyer-Moore theorem prover, i.e. without loading the proof-checker.[12] Actually I still

[12]which is surprising since I'm usually good at proselytizing on behalf of the proof-checker, trying to convince everyone not to bother massaging their proofs so that they'll replay in the raw prover

had problems getting the substitution right below[13], but I went back into the proof-checker again to get it right.

```
(prove-lemma homogeneous1-car-wit-cdr-partition
           (rewrite)
           (homogeneous1
            elt
            (car (wit pairs
                      (cdr (partition elt dom pairs))
                      p q))
            pairs 2)
           ((use (homogeneous1-subset
                  (domain (cdr (partition elt dom pairs)))
                  (flg 2)
                  (x (car (wit pairs
                          (cdr (partition elt dom pairs))
                          p q)))))
            (disable homogeneous1-subset)))
```

Here's the same thing for the `CAR` of `PARTITION`. First, the analogous lemma:

```
(prove-lemma homogeneous1-car-partition (rewrite)
  (homogeneous1 elt (car (partition elt dom pairs)) pairs 1))
```

Now for the analog of the lemma `HOMOGENEOUS1-CAR-WIT-CDR-PARTITION` above.

```
(prove-lemma homogeneous1-car-wit-car-partition
           (rewrite)
           (homogeneous1
            elt
            (car (wit pairs
                      (car (partition elt dom pairs))
                      p q))
            pairs 1)
           ((use (homogeneous1-subset
                  (domain (car (partition elt dom pairs)))
                  (flg 1)
```

[13]By the way, the syntax for Boyer-Moore USE hints is (¡lemma-name¿ {(¡variable¿ ¡term¿)}*).

```
              (x (car (wit pairs
                          (car (partition elt dom pairs))
                          p q)))))
        (disable homogeneous1-subset)))
```

At this point I tried the main goal, WIT-YIELDS-GOOD-HOM-SET, with high hopes. However, it didn't quite go through. Let me explain something that's both a strength and a weakness of a common approach to using the Boyer-Moore theorem prover: submitting theorems to the prover and then trying to get useful feedback from a failed proof. The prover chugged along until it printed the following formula just before its first generalization. (Generalization is a good place to look for needed lemmas, as explained in the prover's user's manual [6].)

```
(IMPLIES
 (AND (NUMBERP W)
      (NUMBERP V)
      (LESSP (DIFFERENCE (LENGTH X)
                         (LENGTH
                          (CDR (PARTITION Z X PAIRS))))
             (RAMSEY V (ADD1 W)))
      (NOT (EQUAL (CDR (WIT PAIRS
                            (CDR (PARTITION Z X PAIRS))
                            (ADD1 V)
                            W))
                  1))
      (HOMOGENEOUS (CAR (WIT PAIRS
                             (CDR (PARTITION Z X PAIRS))
                             (ADD1 V)
                             W))
                   PAIRS
                   (CDR (WIT PAIRS
                             (CDR (PARTITION Z X PAIRS))
                             (ADD1 V)
                             W)))
      (NOT (LESSP (LENGTH
                   (CAR (WIT PAIRS
                             (CDR (PARTITION Z X PAIRS))
                             (ADD1 V)
                             W)))
                  W))
      (NOT (LESSP (LENGTH X)
                  (SUB1 (PLUS (RAMSEY V (ADD1 W))
```

```
                                (RAMSEY (ADD1 V) W))))))
    (HOMOGENEOUS (CAR (WIT PAIRS
                           (CDR (PARTITION Z X PAIRS))
                           (ADD1 V)
                           W))
                 PAIRS 2))
```

This looks pretty awful, but if one weeds out the "garbage" then one may obtain following strengthening. For convenience let's introduce the abbreviations `flg` and `hom-set` for the respective terms

```
  (CDR (WIT PAIRS
            (CDR (PARTITION Z X PAIRS))
            (ADD1 V)
            W))
```

and

```
  (CAR (WIT PAIRS
            (CDR (PARTITION Z X PAIRS))
            (ADD1 V)
            W))
```

So without further ado, here is the resulting simplified formula:

```
   (IMPLIES
    (AND (NOT (EQUAL flg 1))
         (HOMOGENEOUS hom-set PAIRS flg))
    (HOMOGENEOUS hom-set PAIRS 2))
```

If the prover "knew" that the `flg` term is always equal either to `1` or to `2`, then the first hypothesis above would (we expect) cause the second hypothesis to be rewritten to the conclusion! So let's prove the necessary rewrite rule. Interestingly, this failed the first time I tried it, because in fact it wasn't a theorem! At this stage I had a slightly different version of the definition of `WIT`. The problem was that in the case that `DOM` is empty, that definition of `WIT` implied that `(WIT PAIRS DOM P Q)` equals `NIL`. So I went back and fixed the definition of `WIT` to return `(CONS NIL 1)` in that case (which is much more reasonable, i.e. the set returned is `NIL` and the flag doesn't matter but may as well be `1`). Everything replayed successfully in just a very few minutes.

```
(prove-lemma cdr-wit-is-1-or-2 (rewrite)
  (implies (not (equal (cdr (wit pairs dom p q)) 1))
           (equal (cdr (wit pairs dom p q)) 2)))
```

Our goal then went through the prover successfully.

```
(prove-lemma wit-yields-good-hom-set (rewrite)
  (implies (not (lessp (length domain)
                        (ramsey p q)))
           (good-hom-set pairs domain p q
                          (cdr (wit pairs domain p q)))))
```

3.4 The third part of the proof: the witness list represents a set

So, we're about finished! But as was pointed out earlier, SUBSETP is really a notion defined on lists rather than sets. In order to know that cardinality is reasonably represented by LENGTH, we need to prove that the lists in question really have no duplicates. Here again then is the definition of a predicate which says that a given list has no duplicates.

```
(defn setp (x)
  (if (listp x)
      (and (not (member (car x) (cdr x)))
           (setp (cdr x)))
    t))
```

In this subsection we wish to prove:

```
   (prove-lemma setp-hom-set (rewrite)
     (implies (setp domain)
              (setp (car (wit pairs domain p q)))))
```

If we submit this to the theorem prover at this point, though, the following goal appears within a few seconds:

```
  (IMPLIES (AND (LISTP DOMAIN)
```

```
            (NOT (ZEROP P))
            (NOT (ZEROP Q))
            (LESSP (LENGTH (CAR (PARTITION (CAR DOMAIN)
                                           (CDR DOMAIN)
                                           PAIRS)))
                   (RAMSEY (SUB1 P) Q))
            (EQUAL (CDR (WIT PAIRS
                             (CDR (PARTITION (CAR DOMAIN)
                                             (CDR DOMAIN)
                                             PAIRS))
                             P
                             (SUB1 Q)))
                   1)
            (NOT (SETP (CDR (PARTITION (CAR DOMAIN)
                                       (CDR DOMAIN)
                                       PAIRS))))
            (SETP DOMAIN))
       (SETP (CAR (WIT PAIRS DOMAIN P Q))))
```

Trimming it down yields the following strengthening,

```
  (IMPLIES (AND (NOT (SETP (CDR (PARTITION (CAR DOMAIN)
                                           (CDR DOMAIN)
                                           PAIRS))))
                (SETP DOMAIN))
           (SETP (CAR (WIT PAIRS DOMAIN P Q))))
```

— and this clearly has contradictory hypotheses. Thus I proved the following:

```
(prove-lemma setp-partition (rewrite)
  (implies (setp x)
           (and (setp (car (partition a x pairs)))
                (setp (cdr (partition a x pairs))))))
```

At this point I again submitted SETP-HOM-SET to the theorem prover. This time the proof hummed along up to the following goal, which appeared just before the first generalization:

```
  (IMPLIES (AND (NUMBERP W)
                (NUMBERP V)
```

```
          (LESSP (DIFFERENCE
                  (LENGTH X)
                  (LENGTH (CDR (PARTITION Z X PAIRS))))
                 (RAMSEY V (ADD1 W)))
          (NOT (EQUAL (CDR (WIT PAIRS
                                (CDR (PARTITION Z X PAIRS))
                                (ADD1 V)
                                W))
                      1))
          (SETP (CAR (WIT PAIRS
                          (CDR (PARTITION Z X PAIRS))
                          (ADD1 V)
                          W)))
          (NOT (MEMBER Z X))
          (SETP X))
     (NOT (MEMBER Z
                  (CAR (WIT PAIRS
                            (CDR (PARTITION Z X PAIRS))
                            (ADD1 V)
                            W)))))
```

We can trim this down (hence strengthening it) to:

```
(**)  (IMPLIES (NOT (MEMBER Z X))
               (NOT (MEMBER Z
                            (CAR (WIT PAIRS
                                      (CDR (PARTITION Z X PAIRS))
                                      (ADD1 V)
                                      W)))))
```

which is obvious since the `WIT` term is a subset of `X`. There is a lemma MEMBER-SUBSETP which should apply here somehow, but since it has a free variable (cf. page 14) I used the proof-checker to help me to see if this lemma applies. In fact I completed its proof using the proof-checker. For those familiar with the proof-checker, here are the instructions in that proof:

```
promote
(contradict 1)
(rewrite member-subsetp
         (($y
           (car (wit pairs
```

```
                       (cdr (partition z x pairs))
                       (add1 v)
                       w)))))
   (rewrite subsetp-is-transitive
            (($y (cdr (partition z x pairs)))))
   (rewrite subsetp-hom-set-domain)
   (rewrite subsetp-cdr-partition)
```

The dollar signs '$' correspond to substitutions for free variables; notice that there are two of these. Here is the event that I actually submitted to the prover; the USE hints correspond to the instances of the rewrite rules MEMBER-SUBSETP and SUBSETP-IS-TRANSITIVE used in the interactive proof. (One wouldn't expect these to fire automatically, i.e. to apply without needing the USE hint, because of the problem of instantiating the variables occurring in their hypotheses which don't occur in their conclusions.) Actually, the lemma that follows (and that I proved interactively with the instructions shown above) is a bit different from (**) above, since the presence of the `(ADD1 V)` term in (**) made it less general than it should have been.

```
(prove-lemma not-member-car-wit-cdr-partition
       (rewrite)
       (implies (not (member z x))
                (not (member
                      z
                      (car (wit pairs
                                (cdr (partition z x pairs))
                                p
                                q)))))
       ((use (member-subsetp
              (y (car (wit pairs
                           (cdr (partition z x pairs))
                           p
                           q)))
              (x z)
              (z x))
             (subsetp-is-transitive
              (x (car (wit pairs
                           (cdr (partition z x pairs))
                           p
                           q)))
              (y (cdr (partition z x pairs)))
              (z x)))))
```

I tried the proof of our main goal SETP-HOM-SET again but it still didn't go through, because I still needed the following analog for `CAR` of `PARTITION` of the lemma just proved above. (I again used an editor to "write" the following event easily using the one above.)

```
(prove-lemma not-member-car-wit-car-partition
       (rewrite)
       (implies (not (member z x))
                (not (member
                      z
                      (car (wit pairs
                                (car (partition z x pairs))
                                p
                                q)))))
       ((use (member-subsetp
              (y (car (wit pairs
                            (car (partition z x pairs))
                            p
                            q)))
              (x z)
              (z x))
             (subsetp-is-transitive
              (x (car (wit pairs
                            (car (partition z x pairs))
                            p
                            q)))
              (y (car (partition z x pairs)))
              (z x)))))
```

Then I was finally able to get the main goal proved.

```
(prove-lemma setp-hom-set (rewrite)
  (implies (setp domain)
           (setp (car (wit pairs domain p q)))))
```

And now for the final theorem.

```
(prove-lemma ramsey-theorem-2 (rewrite)
  (implies (leq (ramsey p q)
                (length domain))
           (and (subsetp (car (wit pairs domain p q))
```

```
                                domain)
                   (good-hom-set pairs domain p q
                                       (cdr (wit pairs domain p q)))
                   (implies (setp domain)
                                (setp (car (wit pairs domain p q))))))
  ((disable good-hom-set)))
```

References

[1] R.L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

[2] Michael J. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* Volume 78 of *Lecture Notes in Computer Science*, Springer-Verlag, 1979.

[3] Ronald L. Graham, Bruce L. Rothschild, and Joel H. Spencer. *Ramsey Theory.* John Wiley and Sons, 1980.

[4] Matt Kaufmann. [no title]. January 22 1987. (Unpublished).

[5] Matt Kaufmann. *A User's Manual for an Interactive Enhancement to the Boyer-Moore Theorem Prover.* Technical Report CLI-19, Computational Logic, Inc., May 1988.

[6] J Strother Moore Robert S. Boyer. *The User's Manual for A Computational Logic.* Academic Press, Boston, 1988.