A Simple Formalization and Proof for the Mutilated Chess Board

Lawrence C. Paulson
Computer Laboratory, University of Cambridge

April 1996
Revised June 1997

Abstract

The impossibility of tiling the mutilated chess board has been formalized and verified using Isabelle. The formalization is concise because it is expressed using inductive definitions. The proofs are straightforward except for some lemmas concerning finite cardinalities. This exercise is an object lesson in choosing a good formalization.

Contents

1 Introduction

A chess board can be tiled by 32 dominoes, each covering two squares. If two diagonally opposite squares are removed, can the remaining 62 squares be tiled by dominoes? No. Each domino covers a white square and a black square, so a tiled area must have equal numbers of both colours. The mutilated board cannot be tiled because the two removed squares have the same colour.

The mutilated chess board problem has stood as a challenge to the automated reasoning community since McCarthy [9] posed it in 1964. Robinson [17] outlines the history of the problem, citing Max Black as its originator.

Anybody can grasp the argument instantly, but even formalizing the problem seems hard, let alone proving it. McCarthy has recently renewed his challenge, publishing a formalization that he claims is suitable for any "heavy duty set theory" prover [10].

Formalizations like this deny the simplicity of the original problem. They typically define complicated predicates to recognize objects. To recognize dominoes, a predicate checks whether its argument contains two adjacent squares. Subramanian defines "adjacent" by comparing co-ordinates [19, 20]:

```
(defn adjp (s1 s2)
  (or (and (equal (car s1) (car s2)) (equal (plus 1 (cdr s1)) (cdr s2)))
      (and (equal (cdr s1) (cdr s2)) (equal (plus 1 (car s1)) (car s2)))
      ))
```

Subramanian makes other definitions whose combined effect is to recognize a list of non-overlapping dominoes and to compute the region covered. McCarthy's formalization has a similar flavour, though posed in the language of sets. It is concise but formidable.

An alternative is to express the notion of tiling by an inductive definition. It is concise and nearly as clear as the informal problem statement.

2 Mathematical development

First we must make the intuitive argument rigorous. A *tile* is a set, regarded as a set of positions. A *tiling* (using a given set A of tiles) is defined inductively to be either the empty set or the union of a tiling with a tile $a \in A$ disjoint from it. Thus, a tiling is a finite union of disjoint tiles drawn from A.

This view is abstract and general. None of the sets have to be finite; we need not specify what positions are allowed. Now let us focus on chess boards.

A *square* is a pair (i, j) of natural numbers: an *even* (or *white*) square if $i + j$ is even and otherwise an *odd* (or *black*) square.

Let $\mathsf{below}(n) = \{i \mid i < n\}$. (In set theory $n = \{i \mid i < n\}$ by definition, but readers might find this confusing.) The Cartesian product $\mathsf{below}(8) \times \mathsf{below}(8)$ expresses a 64-square chess board; it is the union of 8 disjoint rows of the form $\{i\} \times \mathsf{below}(8)$ for $i = 0, \ldots, 7$.

A *domino* is a tile of the form $\{(i, j), (i, j + 1)\}$ or $\{(i, j), (i + 1, j)\}$. Since tilings are finite, we can use induction to prove that every tiling using dominoes has as many even squares as odd.

Every row of the form $\{i\} \times \mathsf{below}(2n)$ can be tiled using dominoes. As the union of two disjoint tilings is itself a tiling, every matrix of the form $\mathsf{below}(2m) \times \mathsf{below}(2n)$ can be tiled using dominoes. So every $2m \times 2n$ matrix has as many even squares as odd squares. The diagonally opposite squares $(0, 0)$ and $(2m - 1, 2n - 1)$ are both even; removing them results in a set that has fewer even squares than odd squares.[1] No such set, including the mutilated chess board, can be tiled using dominoes.

3 The formal definitions

Isabelle [13] is a generic proof assistant, supporting many logics including ZF set theory and higher-order logic. I have done this exercise using both Isabelle/ZF and Isabelle/HOL. The definitions and proofs are similar in both systems. My formalization should be easy to mechanize in type theory provers, such as Coq [5] and HOL [6], that support inductive definitions. Type theory simplifies the presentation slightly, as type checking eliminates premises such as $i \in \mathsf{nat}$.

Figure 1 presents the Isabelle theory file. Inductive definitions are made implicitly by listing the desired introduction rules. An Isabelle package defines the appropriate least fixedpoint and proves the introduction and induction rules [12]. The set of dominoes is inductively defined by two axioms:

$$\{(i, j), (i, j + 1)\} \in \mathsf{domino} \qquad \{(i, j), (i + 1, j)\} \in \mathsf{domino}$$

Most mathematicians would define dominoes like this instead of reducing the definition to first principles, even though the “induction” here is trivial.

The set of tilings using a set A of tiles is defined inductively too:

$$\emptyset \in \mathsf{tiling}(A) \qquad \frac{a \in A \quad t \in \mathsf{tiling}(A) \quad a \cap t = \emptyset}{a \cup t \in \mathsf{tiling}(A)}$$

[1]This holds only because the board has finitely many squares; recall that there are as many even integers as there are integers.

```
Mutil = Finite +
consts
  domino  :: "(nat*nat)set set"
  tiling  :: "'a set set => 'a set set"
  below   :: "nat => nat set"
  evnodd  :: "[(nat*nat)set, nat] => (nat*nat)set"

inductive "domino"
  intrs
    horiz  "{(i, j), (i, Suc j)} ∈ domino"
    vertl  "{(i, j), (Suc i, j)} ∈ domino"

inductive "tiling A"
  intrs
    empty  "{} ∈ tiling A"
    Un     "[| a ∈ A;  t ∈ tiling A;  a ∩ t = {} |] ⟹ a ∪ t ∈ tiling A"

defs
  below_def  "below n    == {i. i<n}"
  evnodd_def "evnodd A b == A ∩ {(i,j). (i+j) mod 2 = b}"

end
```

Figure 1: Isabelle/HOL Definitions of Dominoes and Tilings

```
goal thy "!!t. t ∈ tiling A ⟹
               u ∈ tiling A → t ∩ u = {} → t ∪ u ∈ tiling A";
by (etac tiling.induct 1);
by (Simp_tac 1);
by (simp_tac (!simpset addsimps [Un_assoc]) 1);
by (blast_tac (!claset addIs tiling.intrs) 1);
```

Figure 2: Isabelle/HOL Proof: the Union of Disjoint Tilings is a Tiling

Here is an example of inductive reasoning from this definition. We shall prove that the union of two disjoint tilings is itself a tiling:

$$\frac{t \in \mathsf{tiling}(A) \quad u \in \mathsf{tiling}(A) \quad t \cap u = \emptyset}{t \cup u \in \mathsf{tiling}(A)}$$

By induction on t there are two cases.

- If $t = \emptyset$ then $t \cup u = u \in \mathsf{tiling}(A)$.
- If $t = a \cup t'$ where $a \cap t' = \emptyset$, then by assumption we have $(a \cup t') \cap u = \emptyset$. So $a \cap u = \emptyset$ and $t' \cap u = \emptyset$. From the latter and the induction hypothesis we have $t' \cup u \in \mathsf{tiling}(A)$. And since a is disjoint from both t' and u, we may add it to this tiling to obtain $a \cup (t' \cup u) \in \mathsf{tiling}(A)$.

Figure 1 contains two simple definitions. The set $\mathsf{below}(n)$ is defined to be $\{i \mid i < n\}$. The set $\mathsf{evnodd}\ A\ b$ denotes the even squares of A if $b = 0$ and the odd squares if $b = 1$; it has an obvious explicit definition using intersection and set comprehension.

4 The mechanical proof

The Isabelle proofs offer few surprises. Finite cardinalities are tricky to reason about, as I have noted in previous work [16]. It took a couple of hours just to prove that a domino consists of one even square and one odd square; a richer library of arithmetic facts would have helped. Another trouble spot was to prove that removing elements from a finite set reduces its cardinality: $|A - \{x\}| < |A|$ if A is finite and $x \in A$.

Apart from these trouble spots, the mechanized proof was straightforward. Developing the original ZF version took under 24 working hours. Excluding facts added to libraries, the (HOL) definitions and proof script occupy under 6400 bytes. They execute in 21 seconds on a SPARC SS-10.

Figure 2 presents part of the script: the inductive proof outlined in the previous section. Isabelle's simplifier and classical reasoner prove the base case and inductive step, respectively.

The development comprises 17 theorems. They include simple facts about $\mathsf{below}(n+1)$ and Cartesian products:

```
below(Suc n) × B  =  ({n} × B) ∪ ((below n) × B)
A × below(Suc n)  =  (A × {n}) ∪ (A × (below n))
```

Any row or matrix with an even number of rows can be tiled with dominoes:

```
      {i} × below(n+n) ∈ tiling domino
(below m) × below(n+n) ∈ tiling domino
```

There are facts about the set $\mathsf{evnodd}\ A\ b$, the squares of a given colour:

```
(i,j) ∈ evnodd A b   =   ((i,j)∈ A  ∧  (i+j) mod 2 = b)
  evnodd (A ∪ B) b   =   evnodd A b ∪ evnodd B b
```

Each domino contains a square of each colour:

```
[| d∈domino; b<2 |] ⟹ ∃ i j. evnodd d b = {(i,j)}
```

Crucial to the cardinality argument is that a tiling by dominoes covers only finitely many squares:

```
t ∈ tiling domino ⟹ finite t
```

Every set tiled by dominoes (such as the full chess board itself) contains as many black squares as white ones.

```
t ∈ tiling domino ⟹ card(evnodd t 0) = card(evnodd t 1)
```

The main result is proved for any board t with positive even dimensions. The mutilated board, t', cannot be tiled with dominoes.

```
[| t = below(Suc m + Suc m) × below(Suc n + Suc n);
   t' = t - {(0,0)} - {(Suc(m+m), Suc(n+n))}
|] ⟹ t' ∉ tiling domino
```

5 Related work

In this note there is no space for a full literature review. Several recent efforts [2, 18, 20] are in the same spirit as the present work: the chess board is formalized and impossibility of tiling proved following the intuitive argument about colours. Other work has used exhaustive search or radical reformulations of the problem.

The Isabelle formalization compares favourably with the others. The definitions are short, and in my view, easy to understand. The script is quite short at just over 200 lines, compared with over 500 for Subramanian [19]. (In terms of characters, which is more accurate, the ratio drops to 1.8.) According to McCarthy [10], Bancerek's mechanization [2] in Mizar requires 400 lines. Rudnicki's version [18] (also in Mizar) requires 300 lines. Andrews [1] reports a complex proof; it is not clear how much effort is needed to generate it.

The inductive definition plays the same role as Subramanian's finite state machine [20]. The initial state is the empty board; next states are obtained by adding disjoint tiles; properties that hold of all reachable states are proved by induction. There are two reasons why the Isabelle mechanization is shorter. Error states need not be formalized: they never occur. And many of the concepts expressed in pure Lisp can be expressed using library notions of Cartesian product, intersection, etc.

The finite state machine approach that Subramanian describes has been applied to substantial system verifications [11]. The inductive approach described above is an effective means of verifying cryptographic protocols [15, 14]. It seems, therefore, that inductive definitions also scale up to serious problems.

6 Conclusions

The mutilated chess board has no practical significance. But it is a telling example of the importance of formalizing the problem in a natural way. Inductive definitions have already demonstrated their worth in proofs about the λ-calculus and programming languages semantics. Their utility here suggests that they will find many further applications.

Acknowledgements. I learned of the expressiveness of inductive definitions through participation in the ESPRIT project 6453 TYPES, and especially through the work of Gérard Huet [7, 8]. John Harrison commented on this paper.

References

[1] P. B. Andrews and M. Bishop. On sets, types, fixed points, and checkerboards. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 1071, pages 1–15. Springer, 1996. 5th International Workshop, TABLEAUX '96.

[2] G. Bancerek. The mutilated chessboard problem — checked by Mizar. In Boyer and Trybulec [3].

[3] R. Boyer and A. Trybulec, editors. *QED Workshop II*. On the World Wide Web at `http://www.mcs.anl.gov/qed/index.html`, 1995.

[4] *10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[5] G. Dowek et al. The Coq proof assistant user's guide. Technical Report 154, INRIA-Rocquencourt, 1993. Version 5.8.

[6] M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge Univ. Press, 1993.

[7] G. Huet. The Gallina specification language : A case study. In *Proceedings of 12th FST/TCS Conference, New Delhi*, LNCS 652. Springer, 1992.

[8] G. Huet. Residual theory in λ-calculus: A formal development. *J. Func. Prog.*, 4(3):371–394, 1994.

[9] J. McCarthy. A tough nut for proof procedures. Memo 16, Stanford Artificial Intelligence Project, July 1964.

[10] J. McCarthy. The mutilated checkerboard in set theory. In Boyer and Trybulec [3].

[11] J. S. Moore. *Piton: A Mechanically Verified Assembly-Level Language*. Kluwer Academic Publishers, 1996.

[12] L. C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In A. Bundy, editor, *Automated Deduction — CADE-12 International Conference*, LNAI 814, pages 148–161. Springer, 1994.

[13] L. C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer, 1994. LNCS 828.

[14] L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In Computer Security Foundations Workshop [4], pages 84–95.

[15] L. C. Paulson. Proving properties of security protocols by induction. In Computer Security Foundations Workshop [4], pages 70–83.

[16] L. C. Paulson and K. Grąbczewski. Mechanizing set theory: Cardinal arithmetic and the axiom of choice. *J. Auto. Reas.*, 17(3):291–323, Dec. 1996.

[17] J. A. Robinson. Formal and informal proofs. In R. S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 267–281. Kluwer Academic Publishers, 1991.

[18] P. Rudnicki. The mutilated checkerboard problem in the lightweight set theory of Mizar. On the World Wide Web at `http://web.cs.ualberta.ca/~piotr/Mizar/Mutcheck`, Nov. 1995.

[19] S. Subramanian. A mechanically checked proof of the mutilated checkerboard theorem. On the World Wide Web at `http://www.cli.com/ftp/pub/nqthm/nqthm-1992/examples/subramanian/`, 1994.

[20] S. Subramanian. An interactive solution to the $n \times n$ mutilated checkerboard problem. *J. Logic and Comput.*, 6(4), 1996. In press.

A Full proof script

```
(*  Title:      HOL/Induct/Mutil
    ID:         $Id: Mutil.ML,v 1.4 1997/06/06 08:46:26 paulson Exp $
    Author:     Lawrence C Paulson, Cambridge University Computer Laboratory
    Copyright   1996  University of Cambridge

The Mutilated Chess Board Problem, formalized inductively
*)

open Mutil;

Addsimps tiling.intrs;

(** The union of two disjoint tilings is a tiling **)

goal thy "!!t. t: tiling A ==> \
\              u: tiling A --> t <= Compl u --> t Un u : tiling A";
by (etac tiling.induct 1);
by (Simp_tac 1);
by (simp_tac (!simpset addsimps [Un_assoc]) 1);
by (blast_tac (!claset addIs tiling.intrs) 1);
qed_spec_mp "tiling_UnI";


(*** Chess boards ***)

goalw thy [below_def] "(i: below k) = (i<k)";
by (Blast_tac 1);
qed "below_less_iff";
AddIffs [below_less_iff];

goalw thy [below_def] "below 0 = {}";
by (Simp_tac 1);
qed "below_0";
Addsimps [below_0];

goalw thy [below_def]
    "below(Suc n) Times B = ({n} Times B) Un ((below n) Times B)";
by (simp_tac (!simpset addsimps [less_Suc_eq]) 1);
by (Blast_tac 1);
qed "Sigma_Suc1";

goalw thy [below_def]
    "A Times below(Suc n) = (A Times {n}) Un (A Times (below n))";
by (simp_tac (!simpset addsimps [less_Suc_eq]) 1);
by (Blast_tac 1);
qed "Sigma_Suc2";
```

```
goal thy "{i} Times below(n+n) : tiling domino";
by (nat_ind_tac "n" 1);
by (ALLGOALS (asm_simp_tac (!simpset addsimps [Un_assoc RS sym, Sigma_Suc2])));
by (resolve_tac tiling.intrs 1);
by (assume_tac 2);
by (subgoal_tac    (*seems the easiest way of turning one to the other*)
    "({i} Times {Suc(n+n)}) Un ({i} Times {n+n}) = \
\    {(i, n+n), (i, Suc(n+n))}" 1);
by (Blast_tac 2);
by (asm_simp_tac (!simpset addsimps [domino.horiz]) 1);
by (Auto_tac());
qed "dominoes_tile_row";

goal thy "(below m) Times below(n+n) : tiling domino";
by (nat_ind_tac "m" 1);
by (ALLGOALS (asm_simp_tac (!simpset addsimps [Sigma_Suc1])));
by (blast_tac (!claset addSIs [tiling_UnI, dominoes_tile_row]
                       addSEs [below_less_iff RS iffD1 RS less_irrefl]) 1);
qed "dominoes_tile_matrix";


(*** Basic properties of evnodd ***)

goalw thy [evnodd_def] "(i,j): evnodd A b = ((i,j): A  &  (i+j) mod 2 = b)";
by (Simp_tac 1);
qed "evnodd_iff";

goalw thy [evnodd_def] "evnodd A b <= A";
by (rtac Int_lower1 1);
qed "evnodd_subset";

(* finite X ==> finite(evnodd X b) *)
bind_thm("finite_evnodd", evnodd_subset RS finite_subset);

goalw thy [evnodd_def] "evnodd (A Un B) b = evnodd A b Un evnodd B b";
by (Blast_tac 1);
qed "evnodd_Un";

goalw thy [evnodd_def] "evnodd (A - B) b = evnodd A b - evnodd B b";
by (Blast_tac 1);
qed "evnodd_Diff";

goalw thy [evnodd_def] "evnodd {} b = {}";
by (Simp_tac 1);
qed "evnodd_empty";

goalw thy [evnodd_def]
    "evnodd (insert (i,j) C) b = \
\      (if (i+j) mod 2 = b then insert (i,j) (evnodd C b) else evnodd C b)";
```

```
by (simp_tac (!simpset setloop (split_tac [expand_if] THEN' Step_tac)) 1);
qed "evnodd_insert";

Addsimps [finite_evnodd, evnodd_Un, evnodd_Diff, evnodd_empty, evnodd_insert];


(*** Dominoes ***)

goal thy "!!d. [| d:domino; b<2 |] ==> EX i j. evnodd d b = {(i,j)}";
by (eresolve_tac [domino.elim] 1);
by (res_inst_tac [("k1", "i+j")] (mod2_cases RS disjE) 2);
by (res_inst_tac [("k1", "i+j")] (mod2_cases RS disjE) 1);
by (REPEAT_FIRST assume_tac);
(*Four similar cases: case (i+j) mod 2 = b, 2#-b, ...*)
by (REPEAT (asm_full_simp_tac (!simpset addsimps [less_Suc_eq, mod_Suc]
                          setloop split_tac [expand_if]) 1
           THEN Blast_tac 1));
qed "domino_singleton";

goal thy "!!d. d:domino ==> finite d";
by (blast_tac (!claset addSEs [domino.elim]) 1);
qed "domino_finite";


(*** Tilings of dominoes ***)

goal thy "!!t. t:tiling domino ==> finite t";
by (eresolve_tac [tiling.induct] 1);
by (rtac Finites.emptyI 1);
by (blast_tac (!claset addSIs [finite_UnI] addIs [domino_finite]) 1);
qed "tiling_domino_finite";

goal thy "!!t. t: tiling domino ==> card(evnodd t 0) = card(evnodd t 1)";
by (eresolve_tac [tiling.induct] 1);
by (simp_tac (!simpset addsimps [evnodd_def]) 1);
by (res_inst_tac [("b1","0")] (domino_singleton RS exE) 1);
by (Simp_tac 2 THEN assume_tac 1);
by (res_inst_tac [("b1","1")] (domino_singleton RS exE) 1);
by (Simp_tac 2 THEN assume_tac 1);
by (Step_tac 1);
by (subgoal_tac "ALL p b. p : evnodd a b --> p ~: evnodd ta b" 1);
by (asm_simp_tac (!simpset addsimps [tiling_domino_finite]) 1);
by (blast_tac (!claset addSDs [evnodd_subset RS subsetD] addEs [equalityE]) 1);
qed "tiling_domino_0_1";

goal thy "!!m n. [| t = below(Suc m + Suc m) Times below(Suc n + Suc n);   \
\                   t' = t - {(0,0)} - {(Suc(m+m), Suc(n+n))}              \
\                |] ==> t' ~: tiling domino";
by (rtac notI 1);
by (dtac tiling_domino_0_1 1);
by (subgoal_tac "card(evnodd t' 0) < card(evnodd t' 1)" 1);
by (Asm_full_simp_tac 1);
by (subgoal_tac "t : tiling domino" 1);
(*Requires a small simpset that won't move the Suc applications*)
by (asm_simp_tac (HOL_ss addsimps [dominoes_tile_matrix]) 2);
by (subgoal_tac "(m+m)+(n+n) = (m+n)+(m+n)" 1);
by (asm_simp_tac (!simpset addsimps add_ac) 2);
by (asm_full_simp_tac
    (!simpset addsimps [mod_less, tiling_domino_0_1 RS sym]) 1);
by (rtac less_trans 1);
by (REPEAT
    (rtac card_Diff 1
     THEN asm_simp_tac (!simpset addsimps [tiling_domino_finite]) 1
     THEN asm_simp_tac (!simpset addsimps [mod_less, evnodd_iff]) 1));
qed "mutil_not_tiling";
```