J Autom Reasoning
DOI 10.1007/s10817-017-9423-7

Fast Formal Proof of the Erdős–Szekeres Conjecture for Convex Polygons with at Most 6 Points

Filip Marić[1]

Received: 1 April 2017 / Accepted: 27 July 2017

Abstract A conjecture originally made by Klein and Szekeres in 1932 (now commonly known as "Erdős–Szekeres" or "Happy Ending" conjecture) claims that for every $m \geq 3$, every set of $2^{m-2} + 1$ points in a general position (none three different points are collinear) contains a convex m-gon. The conjecture has been verified for $m \leq 6$. The case $m = 6$ was solved by Szekeres and Peters and required a huge computer enumeration that took "more than 3000 GHz hours". In this paper we improve the solution in several directions. By changing the problem representation, by employing symmetry-breaking and by using modern SAT solvers, we reduce the proving time to around only a half of an hour on an ordinary PC computer (i.e., our proof requires only around 1 GHz hour). Also, we formalize the proof within the Isabelle/HOL proof assistant, making it significantly more reliable.

Keywords Erdős–Szekeres conjecture · Happy ending problem · Convex polygons · Interactive theorem proving · SAT solving · Isabelle/HOL

1 Introduction

In their seminal paper [8] in 1935 Paul Erdős and George Szekeres proved that for every number $m \geq 3$, there exists a number n (sometimes denoted as $ES(m)$) such that each set of n different points in the plane with no three collinear points contains a convex m-gon. They conjectured that $n = 2^{m-2} + 1$ and this conjecture is called Erdős–Szekeres or the Happy Ending[1] conjecture. More than 80 years later, up to our best knowledge, the conjecture

[1] The name was coined by Paul Erdős, since the original problem posed for $m = 4$ and $n = 5$ by Esther Klein led to her marriage to George Szekeres.

This work has been partially supported by the Grant 174021 of the Ministry of Science of Serbia.

✉ Filip Marić
filip@matf.bg.ac.rs

[1] Faculty of Mathematics, University of Belgrade, Studentski Trg 16, Belgrade, Serbia

Published online: 05 September 2017

remains open, and attracts a lot of research—according to Google Scholar, the original paper [8] was cited more than 1200 times, with increasing number of citations in recent years. Morris and Soltan [19,20] gave two surveys of the recent achievements towards the solution of this problem and some of its close relatives. In 2016 Suk [24] proved the asymptotic variant of the conjecture, showing that $n = 2^{m+o(m)}$.

The conjecture is proved for $3 \leq m \leq 6$, and it is still open for all $m > 6$. The case of triangles ($m = 3$) is trivial, the case of quadrilaterals ($m = 4$) was the original problem solved by Esther Klein, but already the case of pentagons ($m = 5$) was somewhat difficult to prove (according to [8] this case was first proved by E. Makai, but the first published proofs were given in 1970 by Kalbfleisch et al. [13], and in 1974 by Bonnice [5]).

The case of hexagons ($m = 6$) was proved in 2006 by Szekeres and Peters [25]. Their computer proof was based on large and complex enumeration based on a combinatorial model of planar configurations, expressed in terms of signature functions satisfying certain simple necessary conditions. According to authors, the proof took around 1500 h of computing time using cluster computer with processors of up to 2 GHz. The reliability of the proof was increased by B. McKay who independently verified and implemented the same algorithm. Dehnhardt et al. [7] looked for a pen-and-paper proof for hexagons, but have only covered some cases (the ones where the outer convex hull is a pentagon).

In this paper we prove the main results of Szekeres and Peters again, but show how modern automated reasoning tools (SAT solvers and interactive theorem provers), better problem representation and symmetry breaking can make the proof much faster (the fastest variant of the proof for $m = 6$ can be done on a single average PC in less than half an hour) and much more reliable (for $3 \leq m \leq 5$ the proof is fully machine-checked within Isabelle/HOL theorem prover [21], and for $m = 6$ a SAT formula, mechanically shown to be equivalent to the conjecture, is exported from Isabelle/HOL, solved by a SAT solver, and its unsatisfiability proof is checked by a mechanically verified proof checker GRAT [15], all together in under two hours). Therefore, apart for being a small step forward in analyzing the Happy Ending conjecture, we hope that the paper can offer some more general advice on how to approach computer theorem proving of combinatorial conjectures.

All proofs and all programs described in this paper are available online, at http://argo.matf.bg.ac.rs/downloads.

Overview of the paper In Sect. 2 we briefly describe interactive theorem proving and SAT solving. In Sect. 3 we describe an abstract framework for formulating convexity related notions and introduce all definitions necessary to formally state the instances of the Happy Ending conjecture. In Sect. 4 we describe how to encode the instances of the Happy Ending conjecture as instances of the SAT problem. In Sect. 5 we describe the formalization of convex hulls, introduce the notion of nested convex hulls and show how it can be used to optimize the SAT encoding of the Happy Ending conjecture. In Sect. 6 we further improve the SAT encoding by employing symmetry breaking techniques. In Sect. 7 we prove the central correctness theorem that connects the unsatisfiability of the optimized SAT encoding formula with the validity of the corresponding instances of the Happy Ending conjecture. In Sect. 8 we show the experimental results and analyze the time to generate SAT formulas, to prove their unsatisfiability and to verify those results. Finally in Sects. 9 and 10 we describe potential further work and give conclusions.

2 Background

In this section we shall give a brief overview of interactive theorem proving, the Isabelle/HOL proof assistant, the SAT problem and modern SAT solvers.

2.1 Interactive Theorem Proving and Isabelle/HOL

Interactive theorem provers (ITPs, also called *proof-assistants*) are software tools that assist development of formal proofs by human-machine collaboration. They check formal theories specified by their users, ensuring a very high level of reliability. Main application areas are in program verification, security and formalization of mathematics. Many complex mathematical theorems have been recently formally proved and several highly non-trivial software systems have been fully formally verified using proof-assistants. Some of the most successful proof-assistants today are Coq [12], Isabelle/HOL [21], HOL light [10] and many others. Several surveys on interactive theorem proving describe the foundations of ITP technology and the most prominent results in the field [2,9,18].

Isabelle [21] is a generic proof assistant, but its most developed application is higher order logic (Isabelle/HOL). Formalizations of mathematical theories are made by defining new notions (types, constants, functions, etc.), and proving statements about them (lemmas, theorems, etc.). This is often done using the declarative proof language Isabelle/Isar [28]. Isar is a very rich language, and we will here describe only the syntax of constructions used in this paper. Definitions are made using the syntax **definition** `x` **where** `"x = ···"`, where x is the constant being defined. Lemmas are specified using the syntax **`lemma assumes`** *assms* **`shows`** *concl* where *assms* are assumptions and *concl* is the conclusion of the lemma. If there are no assumptions, the keywords **assumes** and **shows** can be omitted (the form **lemma** *concl* can be used). We will also use the syntax **lemma** `"`$\bigwedge x_1, \ldots x_k.\ [\![\ asm_1;\ \ldots;\ asm_n]\!] \Longrightarrow concl$`"` where $asm_1, \ldots, asm_n$ are the assumptions, *concl* is the conclusion, and $x_1, \ldots, x_k$ are universally quantified variables.

Logic formulas are written in the HOL logic using the standard notation (e.g., the connectives $\wedge$, $\vee$, $\longrightarrow$, $\neg$, quantifiers $\forall$ and $\exists$). Terms can use let-bindings (e.g., `let` $x = 3$ `in` $3 * x$) and if-then-else expressions (e.g., `if` $x > 0$ `then` x `else` $-x$), with the standard semantics.

HOL is a typed logic. To express that x is of some type τ we write $x{::}\tau$. The predefined type `bool` denotes Booleans, `nat` denotes natural numbers, `int` denotes integers, and `real` denotes real numbers. The number types support ordinary arithmetic operations (e.g., $+, -, *, /$) and relations (e.g., $<, \leq, =$).

Polymorphic set types are supported. The type τ `set` denotes the type of sets with elements of type τ. Set-theoretic notation is close to that of standard mathematics (with a few minor exceptions). Finite sets are recognized by the predicate `finite`.

Function type is denoted as $\tau_1 \Rightarrow \tau_2$. The functions are usually curried and function applications are written in prefix form, common to functional programming, as $f\ x$ (instead of $f(x)$, that is closer to standard mathematical notation). Anonymous functions are denoted by λ-expressions (e.g., $\lambda\ x.\ x + 1$ is a function with one parameter x, that computes its successor). Currying is used to represent functions of multiple arguments. Functions can be defined by simple definitions (using the keyword **definition**) or by primitive and general recursion (using the keywords **primrec** or **fun**).

Polymorphic list types are also supported. The type τ `list` denotes the type of lists with elements of type τ. Constant lists are denoted by $[x_0, \ldots, x_n]$. N-th element of the list l is denoted by l `!` n, appending two lists is denoted by l_1 `@` l_2, the length of the list

by the function `length`, and the sum of all list elements by the function `listsum`. The higher-order function `map` maps all elements of a list by a given function (yielding a new list containing the images). Given a list of lists, the function `concat` makes a single list containing all elements from the inner lists (it flattens the given list of lists). The function `set` gives a set of elements for the given lists. For a given natural number n, the function `take` takes the first n elements of the given list, while the function `drop` returns the list obtained from the given list by removing its first n elements.

Abstract reasoning is supported by *locales* [3]. A locale is a named context of constants (functions) $f_1, \ldots, f_n$ and assumptions $P_1, \ldots, P_m$ about them that is introduced roughly like **locale** loc = **fixes** $f_1, \ldots, f_n$ **assumes** $P_1, \ldots, P_m$. Locales can be hierarchical as in **locale** $loc = loc_1 + loc_2$ + **fixes** .... In the context of a locale, definitions can be made and theorems can be proved. Locales can be interpreted by concrete instances of $f_1, \ldots, f_n$, and then it must be shown that these satisfy assumptions $P_1, \ldots, P_m$. Theorems proved abstractly within a locale are automatically transferred to all their interpretations.

2.2 SAT Problem and SAT Solvers

SAT problem is the satisfiability problem for propositional logic, concerned with checking if there is an assignment of propositional variables (a valuation, a model) that makes a given propositional formula in conjunctive normal form (CNF) true. SAT is a canonical NP-complete problem and it holds a central place in the field of computational complexity. It has many applications in hardware circuit design and verification, model checking, routing, planning, scheduling etc. Very large SAT formulas can nowadays be checked by very efficient *SAT solvers*.

SAT solvers are complex software, and it is very hard to ensure that they do not contain implementation errors. On the other hand, since they are used in proving properties of hardware, software and even mathematical theorems, the whole proof often relies on their correctness. Several attempts at proving SAT solver correctness have been made [11,16,17,23,30]. However, better results are usually obtained by extending SAT solvers to provide certificates for their results (models for satisfiable and proofs for unsatisfiable formulas). Certificates are checked by independent checkers that are usually much simpler than solvers themselves (or are even formally verified), so they can be trusted with much higher reliability. The state-of-the-art technique for checking SAT certificates is based on the *Deletion Resolution Asymmetric Tautology (DRAT)* proof format and corresponding proof-checking tool DRAT-trim [29]. Recently two new proof formats were developed: *Generalized ResolutIon Trace (GRIT)* [6] and *GRAT* [15]. These enabled efficient proof verification by checkers that are formally verified within proof-assistants (formally verified GRIT checkers are developed in Coq and ACL2, and a formally verified GRAT checker is developed in Isabelle/HOL).

The certificate checking approach enabled integration of SAT solvers into proof-assistant. For example, the proof-assistant Isabelle/HOL that we use in this paper can use the external solver MiniSAT to prove lemmas (by a special tactic called `sat`) [27]. MiniSAT produces unsatisfiability proofs that are then checked by the Isabelle kernel, ensuring correctness. However, up to the best of our knowledge, this integration does not yet use state-of-the-art proof formats (e.g., GRAT) and its efficiency is limited.

A very detailed exposure of the SAT and related problem, successful solvers and applications is given in the Handbook of Satisfiability [4].

3 Abstract Convexity

One of the first questions when solving the Happy Ending conjecture is how to represent the problem, i.e., how to formulate what seems to be a geometric problem as a combinatorial one (that it inherently is). It is well established that convexity and many other notions in computational geometry can be reduced to a *ccw* predicate that checks if three points are oriented counter-clockwise. Knuth gave an excellent exposure of such an approach and gave an axiomatic system that defines properties that an abstract ccw predicate must satisfy (assuming that all the points are in the general position, i.e., that all considered points are distinct and non-collinear) [14]. Sets of points satisfying Knuth's axioms are called *CC systems*. Pichardie and Bertot have used Knuth's axioms to formalize algorithms for computing convex hulls [22], and Szekeres-Peters computer proof of the Happy Ending conjecture for hexagons ($m = 6$) is also performed in a similar framework [25].

We specify two basic predicates (ccw denoting the counter-clockwise orientation and *col* denoting collinearity) within an Isabelle/HOL locale [3]. The type `'p` that denotes points is kept abstract.

```
locale CCSystem =
fixes
   ccw :: "'p ⇒ 'p ⇒ 'p ⇒ bool" and
   col :: "'p ⇒ 'p ⇒ 'p ⇒ bool"
assumes
   ax0: "ccw p q r ⟹ p ≠ q ∧ p ≠ r ∧ q ≠ r ∧ ¬ col p q r" and
   ax1: "ccw p q r ⟹ ccw q r p" and
   ax2: "ccw p q r ⟹ ¬ ccw p r q" and
   ax3: "⟦p = q; p = r; q = r; ¬ col p q r⟧ ⟹
         ccw p q r ∨ ccw p r q" and
   ax4: "⟦ccw t q r; ccw p t r; ccw p q t ⟹ ccw p q r" and
   ax5: "⟦ccw t s p; ccw t s q; ccw t s r; ccw t p q; ccw t q r⟧ ⟹
         ccw t p r" and
   ax5':"⟦ccw s t p; ccw s t q; ccw s t r; ccw t p q; ccw t q r⟧ ⟹
         ccw t p r" and
   col1: "col a b c ⟹ col b a c" and
   col2: "col a b c ⟹ col c a b"
```

Knuth expresses our *ax0* and the precondition to our *ax3* only in prose (he implicitly assumes that all considered points are in a general position). The set of axioms in our locale is not minimal, as not all axioms are independent – for example, Knuth shows that axiom *ax5'* can be derived from axioms *ax1* to *ax5* (but also under the implicit assumption that all considered points are in a general position). We shall show that the axioms are consistent, by giving their interpretation in the Cartesian plane.

$\mathbb{R}^2$ *interpretation* We define the collinearity and orientation in $\mathbb{R}^2$ (the Cartesian plane) and show that they satisfy the CC system axioms. Points are defined as pairs of real numbers (**type_synonym** `point = "real × real"`). The orientation of three points (x_0, y_0), (x_1, y_1) and (x_2, y_2) depends on the following determinant:

$$\begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}$$

In Isabelle/HOL we define the following function (`det3` calculates the determinant).

```
fun "orientation (x0, y0) (x1, y1) (x2, y2) = det3 x0 y0 1 x1 y1 1 x2 y2 1"
```

Both `col` and `ccw` predicates are defined in terms of this orientation function and the sign of its value.

```
definition col :: "'p ⇒ 'p ⇒ 'p ⇒ bool" where
  "col p0 p1 p2 ⟷ orientation p0 p1 p2 = 0"
definition ccw :: "'p ⇒ 'p ⇒ 'p ⇒ bool"
  "ccw p0 p1 p2 ⟷ orientation p0 p1 p2 > 0"
```

Verifying axioms is done by means of locale interpretation [3], and the interpretation proof turns out to be very easy (we followed the exposure given by Pichardie and Bertot [22]), relying on the properties of the field $\mathbb{R}$ (which are given as `field_simps` in Isabelle/HOL).

General position The collinearity predicate `col` enables us to formally introduce the concept of points being in a general position. We shall say that a set of points S is in a general position if it does not contain three distinct collinear points.

```
definition in_general_position :: "'p set ⇒ bool" where
"in_general_position S ⟷
 ¬ (∃ A B C. A ∈ S ∧ B ∈ S ∧ C ∈ S ∧
             A ≠ B ∧ A ≠ C ∧ B ≠ C ∧ col A B C)"
```

Convex polygons The counter-clockwise orientation predicate `ccw` enables us to define the concept of convex polygon.

```
definition convex_polygon :: "'p list ⇒ bool" where
  "convex_polygon p ⟷ length p ≥ 3 ∧
                       (∀ i j k. i < j ∧ j < k ∧ k < length p ⟶
                                 ccw (p ! i) (p ! j) (p ! k))"
```

Although in some cases it might be technically more convenient to extend the previous definition to include an empty list, list with a single point, and a list with two different points, we decided to stick to the classic notion where a polygon must at least be a triangle. Instances of the Happy Ending conjecture usually consider four or more points (the case $m = 3$ is considered trivial), so "polygons" with less than three points are irrelevant.

Note that the previous definition imposes orientation on all triples of points in the list – a definition that considers only orientation of triples that contain two consecutive points from the list would be valid only for simple polygons.

It is easily proved (using *ax0*) that all points in a convex polygon must be different and in a general position.

```
lemma "convex_polygon p ⟹ distinct p"
lemma "convex_polygon p ⟹ in_general_position (set p)"
```

Formal statement of the Happy Ending conjecture The notions introduced so far are the only notions needed to express the Happy Ending conjecture i.e., its specific cases. For example, the case $m = 5$ and $n = 9$ is given by the following theorem.

```
theorem
  assumes "in_general_position S" "card S = 9"
  shows "∃ p. set p ⊆ S ∧ length p = 5 ∧ convex_polygon p"
```

Therefore, the whole formalization relies only on the notions introduced so far, and only they need to be examined carefully. An error in one of the previous definitions could jeopardize the whole formalization effort. On the other hand, all definitions that follow are auxiliary and are relevant only for the proof — an error in the following definitions would not endanger the overall correctness.

4 SAT Encoding

Our main line of attack on the specific instances of the Happy Ending conjecture is to rephrase them as instances of the SAT problem. Therefore, in this section we introduce the basic notions of SAT and then we describe our basic SAT encoding of the Happy Ending conjecture instances.

4.1 Basic SAT Notions

We define SAT formulas in conjunctive normal form (CNF) as lists of clauses, where each clause is a list of literals encoded by integers — positive literals (variables) are encoded by positive integers, negative literals (negated variables) by corresponding negative integers, and zero is not used. The semantics is given by valuations that map variables (natural numbers) to Boolean values, and is defined in quite a standard manner.

```
definition satisfies_lit :: "(nat ⇒ bool) ⇒ int ⇒ bool" where
  "satisfies_lit val lit ⟷
   (if lit > 0 then val (nat lit) else ¬ val (nat (-lit)))"
definition satisfies_clause :: "(nat ⇒ bool) ⇒ int list ⇒ bool" where
  "satisfies_clause val cl ⟷ (∃ lit ∈ set cl. satisfies_lit val lit)"
definition satisfies_formula :: "(nat ⇒ bool) ⇒ int list list ⇒ bool" where
  "satisfies_formula val fml ⟷ (∀ cl ∈ set fml. satisfies_clause val cl)"
```

We shall use a shorter notation `val ⊨ lit` to denote `satisfies_lit val lit` and `val ⊭ lit` to denote `¬ satisfies_lit val lit`. The same notation shall also be used for the satisfaction of formulas (e.g., `val ⊨ fml`).

Since some formulas that we shall work with contain many unit clauses (clauses containing just one literal), we define the following simplification procedure: unit clauses are kept, but all longer clauses that contain those unit literals are deleted, and opposites of unit literals are deleted from the remaining clauses.

```
definition simplify_formula :: "int list list ⇒ int list list" where
  "simplify_formula fml =
    (let units = filter (λ c. length cl = 1) fml;
         clauses = filter (list_all (λ l. [l] ∉ set units)) fml
     in units @ map (filter (λ l. [l] ∉ set units)) clauses)"
```

Since there are no zeroes in the formula, the simplified formula is equivalent to the original one.

```
lemma
  assumes "lits_not_zero formula"
  shows "val ⊨ fml ⟷ val ⊨ (simplify_formula fml)"
```

4.2 Encoding the Happy Ending Problem

Our SAT encoding of the Happy Ending problem goes as follows. We construct a SAT formula that encodes that the set of n points in general position satisfies the counter-clockwise orientation axioms (*ax1*-*ax5*) and that there is no convex m-gon among those points. Then we prove that this formula is unsatisfiable for $3 \leq m \leq 6$ and corresponding n obtained from the Happy Ending conjecture (e.g., $m = 5, n = 9$, or $m = 6, n = 17$). The unsatisfiability of the formula implies that all points that satisfy the orientation axioms must contain a convex m-gon. Since the standard counter-clockwise orientation predicate for points in $\mathbb{R}^2$ satisfies the orientation axioms, the formula also implies that the given case of the Happy

Ending conjecture holds for points in $\mathbb{R}^2$. However, note that the potential satisfiability of the formula need not imply that the Happy Ending conjecture fails. Namely, as Knuth notes [14], it is possible that all axioms are satisfied, but that such orientation cannot be realized by the points in $\mathbb{R}^2$ (a CC system can be unrealizable).

4.3 Variables and Literals

We use Boolean variables to encode the orientation of triples of points. Given a finite set of points, it is always possible to order it somehow and to assign ordinal numbers, i.e., numeric indices (natural numbers from 0 to $n-1$) to points. The SAT encoding shall only use those indices, and no other properties of points. For each triple of point indices `p < q < r < n`, a separate Boolean variable is introduced, representing the orientation of the corresponding points (the variable shall be true if and only if the triple of points corresponding to indices `p`, `q`, `r` is oriented counter-clockwise). Given a triple of different indices `p`, `q`, and `r`, corresponding to the points p, q, and r, six possible `ccw` predicates can be formed (`ccw` $p\ q\ r$, `ccw` $p\ r\ q$, `ccw` $q\ p\ r$, `ccw` $q\ r\ p$, `ccw` $r\ p\ q$, and `ccw` $r\ q\ p$). All these shall be encoded by literals formed from a single Boolean variable and an appropriately determined sign. Axioms *ax1-ax3* shall be encoded only implicitly through the literals, while axioms *ax4* and *ax5* shall be encoded explicitly as sets of clauses.

Let `ccw_var` be a function that maps indices `p < q < r < n` into the number of the Boolean variable which encodes that points that correspond to those indices are oriented counter-clockwise. Note that it is convenient that the `ccw_var` function also receives the total number of points `n`. Since we are not yet interested in the technicalities of the `ccw_var` definition, we specify it abstractly in the following locale and the proofs that follow use only the two properties given in the locale.

```
locale ccw_var =
  fixes ccw_var :: "nat ⇒ nat ⇒ nat ⇒ nat ⇒ nat"
  assumes ccw_var_inj: "⟦p1 < q1; q1 < r1; r1 < n; p2 < q2; q2 < r2; r2 < n;
                         ccw_var n p1 q1 r1 = ccw_var n p2 q2 r2⟧ ⟹
                         p1 = p2 ∧ q1 = q2 ∧ r1 = r2"
  assumes ccw_var_gt0: "⟦p < q; q < r; r < n⟧ ⟹ ccw_var n p q r > 0"
```

One possible implementation of this function is given by the following, rather non-interesting, technical, definition.

```
definition ccw_var :: "nat ⇒ nat ⇒ nat ⇒ nat ⇒ nat" where
  "ccw_var n p q r =
   listsum (map (λ m. ((n - m) * (n - m - 1)) div 2) [1 ..< p+1]) +
           (q - p - 1) * (2 * n - q - p - 2) div 2 +
           r - q"
```

The following functions are used to get the literal that encodes that points corresponding to indices `p`, `q`, and `r` are counter-clockwise oriented (note that in this case the indices need not be sorted in advance). To find such literal, indices are first sorted using the insertion sort algorithm, the `ccw_var` function is used to get the variable for the sorted triple, and the sign is determined by the number of inversions made during sorting.

```
definition sort_triple :: "nat ⇒ nat ⇒ nat ⇒
                           nat × nat × nat × bool" where
  "sort_triple p q r =
    (let pos = True;
         (p, q, pos) = if p < q then (p, q, pos) else (q, p, ¬ pos);
```

```
          (q, r, pos) = if q < r then (q, r, pos) else (r, q, ¬ pos);
          (p, q, pos) = if p < q then (p, q, pos) else (q, p, ¬ pos)
       in (p, q, r, pos))"

definition ccw_lit :: "nat ⇒ nat ⇒ nat ⇒ nat ⇒ int" where
   "ccw_lit n p q r =
     (let (p, q, r, pos) = sort_triple p q r;
          var = ccw_var n p q r;
       in if pos then (int var) else -(int var)"
```

4.4 The Basic Formula

We shall now describe SAT formulas that encode that points satisfy axioms *ax4* and *ax5* and that there is no convex m-gon among them.

Axiom 4 Although *ax4* holds for all quadruples of points (including all their permutations), there is no need to include all those clauses in the formula. As Knuth notes [14], it suffices to consider only ordered quadruples (where `p < q < r < t`), and for each such quadruple it suffices to include only two clauses. This leads to the following definition of the function that builds the formula encoding that `n` points satisfy *ax4*.

```
definition formula_ax4 :: "nat ⇒ int list list" where
"formula_ax4 n =
 concat (map (λ p. concat (map (λ q. concat (map (λ r. concat (map (λ t.
 [[-ccw_lit n t q r, -ccw_lit n p t r, -ccw_lit n p q t, ccw_lit n p q r],
  [-ccw_lit n q t r, -ccw_lit n t p r, -ccw_lit n q p t, ccw_lit n q p r]
 ]) [r+1..<n])) [q+1..<n])) [p+1..<n])) [0..<n])"
```

The number of clauses in the previous formula is $2 \cdot \binom{n}{4}$, and each clause contains 4 literals. For $n = 17$ this gives 4 760 clauses with the total of 19 040 literals. Roughly it could be said that there are $O(n^4)$ clauses with total $O(n^4)$ literals in this formula.

This definition is given in executable terms (provided that `ccw_var` is made executable). However, in proofs it is much better to have a more abstract characterization.

```
lemma "val ⊨ (formula_ax4 n) ⟷
       (∀ p q r t. {p, q, r, t} ⊆ set [0..<n] ∧ distinct [p, q, r, t] ⟶
                   (val ⊨ ccw_lit n t q r ∧ val ⊨ ccw_lit n p t r ∧
                    val ⊨ ccw_lit n p q t ⟶ val ⊨ ccw_lit n p q r)"
```

The previous lemma proves that if the set of clauses used in `formula_ax4` is satisfied, then all the conditions corresponding to *ax4* (conditions corresponding to arbitrary choice of points `p`, `q`, `r`, and `t` among `n` points) are also satisfied. However, although this also gives us completeness in some sense, i.e., some certainty that we have included all the necessary clauses in our formula, this is not used in the overall correctness proof.

Axiom 5 We define the formula for *ax5* in a very similar fashion. Again not all 5-tuples of points need not be considered — it suffices to assume that `p < q < r`, and that `t` and `s` are different from `p`, `q`, and `r`, and different among themselves. For each such choice, again, only two clauses need to be included in the formula (in the following definition, `list_diff` denotes the difference between the two lists i.e., the list obtained by removing all members of the second list from the first one).

```
definition formula_ax5 :: "nat ⇒ int list list" where
"formula_ax5 n =
```

```
  concat (map (λ p. concat (map (λ q. concat (map (λ r.
    concat (map (λ s. concat (map (λ t.
  [[-ccw_lit n t s p, -ccw_lit n t s q, -ccw_lit n t s r,
    -ccw_lit n t p q, -ccw_lit n t q r, ccw_lit n t p r],
   [-ccw_lit n t s p, -ccw_lit n t s q, -ccw_lit n t s r,
    -ccw_lit n t q p, -ccw_lit n t r q, ccw_lit n t r p]
  ])
     (list_diff [0..<n] [p, q, r, s]))) (list_diff [0..<n] [p, q, r])))
  ([q+1..<n]))) [p+1..<n])) [0..<n])"
```

The number of clauses in the previous formula is $2 \cdot \binom{n}{3} \cdot (n-3) \cdot (n-4)$, and each clause contains 5 literals. For $n = 17$ this gives 247 520 clauses with the total of 1 485 120 literals. Roughly it could be said that there are $O(n^5)$ clauses with total $O(n^5)$ literals in this formula.

Again, it is convenient to have a more abstract characterization of this formula. The following characterization proves that our choice of clauses is complete (and again, this is not used in the overall correctness proof).

```
lemma "val ⊨ (formula_ax5 n) ⟷
       (∀ p q r s t. {p,q,r,s,t} ⊆ set [0..<n] ∧ distinct [p, q, r, s, t] ⟶
                     (val ⊨ ccw_lit n t s p ∧ val ⊨ ccw_lit n t s q ∧
                      val ⊨ ccw_lit n t s r ∧ val ⊨ ccw_lit n t p q ∧
                      val ⊨ ccw_lit n t q r ⟶ val ccw_lit n t p r))"
```

No convex m-gon Finally, we generate a formula expressing that there is no convex m-gon in the given set of n points. Each m-tuple of different points could potentially form a convex m-gon, so it seems that all variations without repetition of m among n elements should be considered. However, due to invariance of convex polygons under cyclic permutations, among m chosen points we can always choose the one with the minimal index to be the starting point of the polygon. Therefore, we need not consider all variations, but just the ones in which the first element is the minimal. We formulate the predicate `min_hd l` that expresses this property (the head of the list is its minimal element), and implement the function `variations_min_hd m n` that builds a list of all variations with the minimal head. The definition of that function is recursive, and rather technical, but very straightforward, so we omit it from the presentation. Once the m points of the polygon are selected, we impose the orientation to all its ordered triples of points (following the definition of the convex polygon).

```
definition formula_no_polygon :: "nat ⇒ nat ⇒ int list list" where
"formula_no_polygon m n =
  map (λ p. concat (map (λ k. concat (map (λ j. map (λ i.
  -ccw_lit n (p ! i) (p ! j) (p ! k)) [0..<j]) [1..<k])) [2..<m]))
      (variations_min_hd m n)"
```

This formula is by far the largest and it dominates the whole encoding. The number of clauses is equal to the number of variations with the minimal head which is $\sum_{i=1}^{n-m+1} \prod_{j=n-i}^{n-m+2-i} j$. For $m = 6$ and $n = 17$, the number of clauses is 1 485 120. Each clause contains $\binom{m}{3}$ literals. For $m = 6$ that is 20 literals, giving the total of 29 702 400 literals. Roughly it could be said that there are $O(n^m)$ clauses with total $O(n^m \cdot m^3)$ literals in this formula.

Again, we can give a more abstract characterization of this formula, and prove that our choice of clauses is complete (and again, that fact is not used in the overall correctness proof).

```
lemma "val ⊨ (formula_no_polygon m n) ⟷
       (∀ p. length p = m ∧ set p ⊆ set [0..<n] ∧ distinct p ⟶
             (∃ i j k. i < j ∧ j < k ∧ k < length p ∧
                       val ⊭ ccw_lit n (p ! i) (p ! j) (p ! k)))"
```

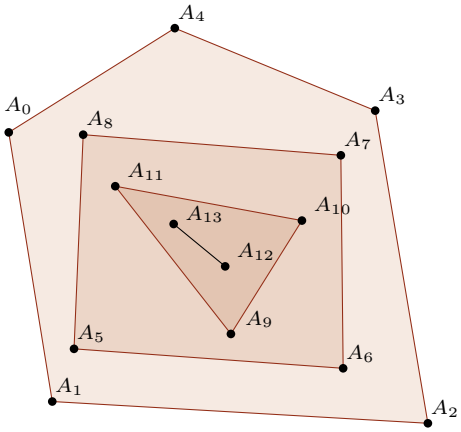


Fig. 1 Nested convex hulls with the structure [5, 4, 3, 2]

The basic formula The basic formula encoding the Happy Ending conjecture for the given `m` and `n` is the conjunction of `formula_ax4 n`, `formula_ax5 n` and `formula_no_polygon m n`. The central correctness theorem claims that if that formula is unsatisfiable, then the Happy Ending conjecture holds for the given `m` and `n`. However, we postpone the proof until Sect. 7, as we shall need to make further modifications in order to optimize this basic formula.

5 Convex Hulls and Their Nesting

The basic SAT encoding formula we have just described is unsatisfiable for all known values of Happy Ending pairs (for $(m, n) \in \{(3, 3), (4, 5), (5, 9), (6, 17)\}$). However, for (6, 17) it is very hard to show unsatisfiability even by the most advanced SAT solvers, so we need to make further optimizations.

Our main optimization technique is to organize the set of points into a list of nested convex hulls (as introduced by Bonnice [5] in his pen-and-paper proof of the Happy Ending problem for pentagons). That list starts with the convex hull of the whole set, continues with the convex hull of the remaining points (when the points of the outer convex hull are removed), and so on, until no points remain. Figure 1 illustrates this concept. Innermost hull can be a convex polygon, but also either a single point or a segment.

Instead of a single formula, we shall consider several separate formulas, each fixing one possible nesting of convex hulls, by extending the basic formula by clauses that impose the specific structure of nested convex hulls. Fixing this structure determines the orientation of many triples in advance and significantly simplifies the formula, so we break one big and hard SAT problem to several easier ones. Additionally, knowing the nesting of convex hulls helps us to add symmetry breaking conditions (stemming from possible cyclic permutations of each nested hull) that further significantly speed up the SAT solving process.

Therefore, we needed to formalize convex hulls and their nesting, and this is described in the rest of this section.

5.1 Convex Hulls

In this section we formally introduce *convex hulls* of sets of points, again vaguely following Knuth and Pichardie and Bertot [14,22]. The convex hull is empty only if the set is empty.

Otherwise it is a list containing different elements from S, that encompasses all elements from S, i.e., it is a list such that for all its pairs of consecutive points s and t (including the pair formed by the last and the first point in the list), for all other points in $r \in S$, the orientation of t, s, r is counter-clockwise. The following predicate checks if a given list of points is a convex hull for the given set of points.

```
definition convex_hull :: "'p list ⇒ 'p set ⇒ bool" where
  "convex_hull p S ⟷
   (S ≠ {} ⟶ p ≠ []) ∧ set p ⊆ S ∧ distinct p ∧
   (let n = length p
     in ∀ i. 0 ≤ i ∧ i < n ⟶
             (let s = p ! i;
                  t = p ! ((i + 1) mod n)
              in (∀ r ∈ S. r ≠ s ∧ r ≠ t → ccw s t r)))"
```

Convex hulls for sets with less than three elements are trivially characterized.

```
lemma "convex_hull [] S ⟷ S = {}"
lemma "convex_hull p {} ⟷ p = []"
lemma "convex_hull p {A} ⟷ p = [A]"
lemma "A ≠ B ⟹ convex_hull p {A, B} ⟷ p = [A, B] ∨ p = [B, A]"
lemma "⟦finite S; card S < 3; convex_hull p S⟧ ⟹ length p = card S"
lemma "⟦card S ≥ 3; convex_hull p S⟧ ⟹ length p ≥ 3"
```

The convex hull of a convex polygon is that polygon itself.

```
lemma "convex_polygon p ⟹ convex_hull p (set p)"
```

The proof that the convex hull of a set with more than three points in a general position is always a convex polygon is more involved, but its formalization was not too hard since an informal proof was described by Knuth.

```
lemma
  assumes "in_general_position S" "convex_hull p S" "length p ≥ 3"
  shows "convex_polygon p"
```

Finally, we needed to show that every finite set of points in a general position always has a convex hull.

```
lemma
  assumes "finite S" "in_general_position S"
  shows "∃ p. convex_hull p S"
```

It turned out that this property was not at all easy to prove, as it required a constructive proof. We needed to specify the incremental algorithm for computing convex hulls and prove its correctness. Although both Knuth and Pichardie and Bertot outline this proof, formalization in Isabelle required some non-trivial effort.

5.2 Nested Convex Hulls

We formally define the notion of nested convex hulls inductively. The empty list represents nested convex hulls for the empty set of points. To a given list of nested convex hulls `H` we can prepend a nonempty list `p`, disjoint from the points in `H`, that is a convex hull of all the points both in `p` and `H`.

```
inductive nested_hulls :: "'p list list ⇒ bool" where
    base: "nested_hulls []"
|   ind: "⟦nested_hulls H; p ≠ []; set p ∩ set (concat H) = {};
           convex_hull p (set p ∪ set (concat H))⟧ ⟹
           nested_hulls (p # H)"
```

Every finite set of points in a general position has a list of nested convex hulls.

```
lemma
  assumes "finite S" "in_general_position S"
  shows "∃ H. nested_hulls H ∧ set (concat H) = S"
```

The proof is very simple, by total induction on the cardinality of `S`, and relies on the previously proved existence of a convex hull for every set of points in a general position.

Structure of nested convex hulls A set of n points can have different structure i.e., the cardinality of elements in the list of nested convex hulls. For example, nested convex hulls in Fig. 1 have the structure [5, 4, 3, 2], meaning that the outer hull is a pentagon, containing a quadrilateral, containing a triangle, containing a segment. Given a list of nested hulls `H`, its hull structure `Hs` is given by `map length H`.

If nested convex hulls contain an element of length $m \geq 3$, since that element is a convex polygon, the original set of n points contains a convex m-gon. Therefore, if a set of n points does not contain a convex m-gon, all elements in its nested convex hulls must have less than m elements (as each convex polygon with more than m elements contains a convex polygon with exactly m elements). The following recursive function computes all possible structures of nested convex hulls for the set of n points that do not contain a convex m-gon.

```
fun nested_hulls_structure :: "nat ⇒ nat ⇒ nat list list" where
  "nested_hulls_structure m n =
    (if n = 0 then [[]]
     else if n < 3 then [[n]]
     else concat (map (λ k.
          if n ≥ k then
               map (op # k) (nested_hulls_structure m (n-k))
          else
               []) [3..<m]))"
```

This function precisely characterizes all possible variants of the structure of nested convex hulls for n points in a general position with no m-gon within nested convex hulls, i.e., the following lemma holds.

```
lemma
  assumes
   "nested_hulls H"
   "finite (set (concat H))" "card (set (concat H)) = n"
   "in_general_position (set (concat H))" "m ≥ 3"
   "¬ (∃ p. set p ⊆ set (concat H) ∧ length p = m ∧ convex_polygon p)"
  shows
   "map length H ∈ set (nested_hulls_structure m n)"
```

The only possible structure of nested convex hulls for $n = 5$ points with no convex hull containing a quadrilateral ($m = 4$) is [3, 2]. Possible structure of nested convex hulls for $n = 9$ points with no convex hull containing a pentagon ($m = 5$) can be [4, 4, 1], [4, 3, 2], [3, 4, 2], and [3, 3, 3]. For $n = 17$ and $m = 6$ there are 73 different possibilities (listed in Table 2). Interestingly, when doing pen-and-paper analysis, Bonnice found only 70 different possibilities [5], and Dehnhardt et al. "corrected" this to 72 [7]. This indicates the importance of formalization and machine-checked proofs.

SAT encoding of the convex hull nesting When defining the basic encoding, we already assumed that points are ordered somehow and that each point has a unique numeric index (from 0 to $n-1$). Now we shall also assume that the order of points is such that the points in each nested convex hull have consecutive indices, and that the numbering goes inwards (points in the outermost i.e., the first convex hull shall have the lowest numbers). The numbering of points in Fig. 1 satisfies this criterion. Suppose that the structure of nested convex hulls of a set of points is given by a list of numbers `Hs`. Then the hull structure `Hs` uniquely determines the indices of points in the hulls (e.g., if `Hs` is `[4, 3, 2]`, the indices of points in the hulls are `[[0, 1, 2, 3], [4, 5, 6], [7, 8]]`. The index `start` of the first point in the n-th hull can be calculated as the sum of previous elements of `Hs`. The indices of all points in that hull are `start`, `start + 1`, ..., `start + Hs!n - 1`.

We shall now formulate two SAT formulas that impose orientation for triples of points, enforced by the given structure of nested convex hulls. The first formula shall be based on the fact that each nested hull with more than two points is a convex polygon, and the second one on the fact that each nested hull encompasses all points in its subsequent hulls.

Given a set of points in a general position, the first formula encodes that all elements of their nested convex hulls that have three or more points are convex polygons (only the last element can be a list with one ore two points, that, by our definition, is not a convex polygon).

```
lemma
  assumes "nested_hulls H" "in_general_position (set (concat H))"
          "p ∈ set H" "length p ≥ 3"
  shows "convex_polygon p"
```

Therefore, all triples of points in all nested convex hulls are adequately oriented.

```
lemma
 assumes
  "nested_hulls H"
  "in_general_position (set (concat H))" "p ∈ set H"
 shows "∀ i j k. i < j ∧ j < k ∧ k < length p ⟶
                  ccw (p ! i) (p ! j) (p ! k)"
```

When points are represented by their numeric indices, and when the structure of the convex hulls is determined by a list `Hs` the previous condition can be encoded by the following SAT formula.

```
definition formula_nested_hulls_polygons ::
           "nat list ⇒ int list list" where
"formula_nested_hulls_polygons Hs =
  concat (map (λ n. let N = listsum Hs; start = listsum (take n Hs);
                    in concat (map (λ k. concat (map (λ j. map (λ i.
                       [ccw_lit N (start + i) (start + j) (start + k)])
               [0..<j]) [1..<k])) [2..<Hs!n])) [0..<length Hs])"
```

For example, when `Hs` is `[4, 3, 2]`, `N` is `9` and `n` takes values `0`, `1`, and `2`. For `n=0`, `start` is `0`, and the clauses `[ccw_lit 9 0 1 2]`, `[ccw_lit 9 0 1 3]`, `[ccw_lit 9 0 2 3]`, and `[ccw_lit 9 1 2 3]` describe that the points `0`, `1`, `2`, and `3` form a convex polygon (`k` ranges from `2` to `3`, `j` from `1` to `k-1`, and `i` from `0` to `j-1`). For `n=1`, `start` is `4`, and the clause `[ccw_lit 9 4 5 6]` describes that points corresponding `4`, `5`, and `6` form a convex polygon. For `n=2`, `start` is `7`, but no clauses are generated (`k` takes values from the empty interval `[2..<2]`, since points `7` and `8` form a segment, and not a polygon).

Note that this formula consists only of unit clauses and these can significantly reduce the size of the final formula during simplification.

As before, we can easily prove an abstract characterization of this formula.

```
lemma "val ⊨ (formula_nested_hulls_polygons Hs) ⟶
       (∀ n < length Hs.
       let N = listsum Hs; start = listsum (take n Hs)
       in ∀ i j k. i < j ∧ j < k ∧ k < Hs!n ⟶
                val ⊨ ccw_lit N (start + i) (start + j) (start + k))"
```

The second formula encodes that each nested hull encompasses all points from its subsequent hulls.

```
lemma
  assumes "nested_hulls H" "n < length H"
  shows "∀ i < length (H ! n).
         let s = H ! n ! i;
             t = H ! n ! ((i + 1) mod length (H ! n))
         in (∀ r ∈ set (concat (drop (n + 1) H)). ccw s t r)"
```

Again, when points are represented by their numeric indices, and when the structure of the convex hulls is determined by a list `Hs` the previous condition can be encoded by the following SAT formula.

```
definition formula_nested_hulls_nested ::
           "nat list ⇒ int list list" where
"formula_nested_hulls_nested Hs =
  concat (map (λ n. let N = listsum Hs; start = listsum (take n Hs)
                    in concat (map (λ i. map (λ x.
               [ccw_lit N (start + i) (start + (i + 1) mod Hs!n) x])
               [start + Hs!n..<N]) [0..<Hs!n]))) [0..<length Hs - 1]))"
```

For example, when `Hs` is `[4, 3, 2]`, `N` is 9 and `n` takes values 0 and 1. For `n=0`, `start` is 0, `i` takes the values 0, 1, 2, and 3, and `x` ranges from 4 to 8. The clauses `[ccw_lit 9 0 1 4]`, `[ccw_lit 9 1 2 4]`, `[ccw_lit 9 2 3 4]`, and `[ccw_lit 9 3 0 4]` force that the point 4 is inside the quadruple formed by the points 0, 1, 2, and 3. Similar clauses that force that points 5, 6, 7, and 8 are inside that quadruple are also formed. For `n=1`, `start` is 4, `i` takes the values 0, 1, and 2 and `x` takes the values 7 and 8. The clauses `[ccw_lit 9 4 5 7]`, `[ccw_lit 9 5 6 7]`, `[ccw_lit 9 6 4 7]` force that the point 7 is inside the triangle formed by points 4, 5, and 6. Similar clauses that force that the point 8 is inside that triangle are also formed.

This formula also consists solely of unit clauses that significantly simplify the final formula and reduce the search space.

An abstract characterization of this formula is the following.

```
lemma "val ⊨ (formula_nested_hulls_nested Hs) ⟷
   (∀ n < length Hs - 1.
      let N = listsum H; start = listsum (take n Hs)
       in ∀ i < Hs!n. ∀ x. x ≥ start + Hs!n ∧ x < listsum Hs ⟶
            val ⊨ ccw_lit N (start + i) (start + ((i + 1) mod Hs!n)) x)"
```

6 Symmetry Breaking

Now we shall show how to further optimize the SAT formula by adding additional symmetry breaking clauses.

6.1 Cyclic Permutations of Polygons

Every list of points forming a convex polygon can be permuted (rotated) cyclically, yielding essentially the same convex polygon. We shall often use such rotations they shall be our central technique for symmetry breaking. We formalize this using the notion of cyclic permutation of a list of points.

```
definition cyclic_perm :: "'p list ⇒ 'p list ⇒ bool" where
 "cyclic_perm p p' ⟷
  p = p' ∨ (∃ n < length p. p' = drop n p @ take n p)"
```

Although it seems that in the previous definition the first case `p = p'` is covered when `n` is zero in the second case, that does not hold when `p` is empty, so this special first case had to be included. The relation `cyclic_perm` is an equivalence (reflexive, symmetric, and transitive) and it preserves convex polygons, and convex hulls.

```
lemma
  assumes "cyclic_perm p p'"
  shows "convex_polygon p ⟷ convex_polygon p'"
lemma
  assumes "cyclic_perm p p'"
  shows "convex_hull p S ⟷ convex_hull p' S"
```

In fact, convex hulls are unique up to cyclic permutations, but we did not need and did not prove this property.

All elements of nested convex hulls can also be cyclically permuted, retaining nested convex hulls.

```
lemma
  assumes "length H' = length H"
          "∀ i < length H. cyclic_perm (H ! i) (H' ! i)"
  shows "nested_hulls H ⟷ nested_hulls H'"
```

If $m \leq 6$ and if there is no convex m-gon in the set of points, then all nested convex hulls are pentagons, quadrilaterals or triangles (except the innermost hull that can consist of one or two points). In the following subsections we shall consider those three cases separately and show how to apply symmetry breaking in each of those three cases. Note that we do not claim that we shall remove all symmetries — further research might find other symmetries that can cause additional search space pruning. However, the symmetries that we shall recognize and remove turn out to be effective enough to prove the $m = 6$ case of the Happy Ending conjecture quite easily.

6.2 Symmetry Breaking for Quadrilaterals

Consider a convex hull that is nonempty quadrilateral `pqrs`. Let `x` be a point within it. The diagonals `pr` and `qs` of the quadrilateral divide it into four different areas (see Fig. 2). The area that contains `x` is determined by the orientation of triples `prx` and `qsx`.

```
definition "area_4_1 p q r s x ⟷ ¬ ccw p r x ∧ ccw q s x"
definition "area_4_2 p q r s x ⟷ ¬ ccw p r x ∧ ¬ ccw q s x"
definition "area_4_3 p q r s x ⟷ ccw p r x ∧ ¬ ccw q s x"
definition "area_4_4 p q r s x ⟷ ccw p r x ∧ ccw q s x"
```

All four areas are mutually symmetric and the points of the quadrilateral can be renumbered (cyclically permuted) so that any given point comes to the area 1.

Fig. 2 Areas of a quadrilateral

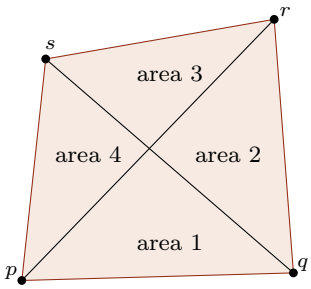
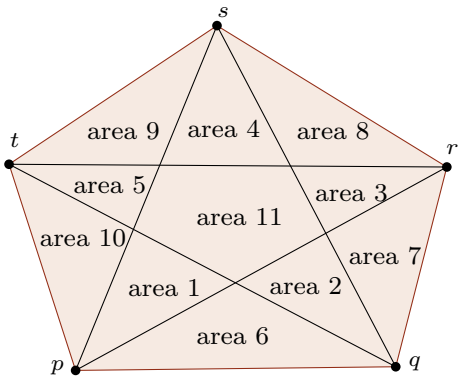


Fig. 3 Areas of a pentagon




```
lemma
  assumes
   "in_general_position {p, q, r, s, x}"
  "convex_polygon [p, q, r, s]"
    "ccw p q x" "ccw q r x" "ccw r s x" "ccw s p x"
shows "∃ p' q' r' s'. cyclic_perm [p, q, r, s] [p', q', r', s'] ∧
                      area_4_1 p' q' r' s' x"
```

The previous lemma holds for any point inside the quadrilateral. Given a list of nested convex hulls, we shall rotate every nonempty nested convex hull that is a quadrilateral so that the first point of the next (contained) convex hull lies in the quadrilateral's area 1. We shall say that all quadrilaterals in nested convex hulls `H` are in canonical positions if the following holds.

```
definition nested_hulls_4_canon :: "'p list list ⇒ bool" where
  "nested_hulls_4_canon H ⟷
   (∀ i < length H-1. length (H!i) = 4 ⟶
      area_4_1 (H!i!0) (H!i!1) (H!i!2) (H!i!3) (H!(i+1)!0))"
```

It can be shown that this can always be achieved, i.e., that for every list of nested convex hulls `H` there is another list of nested convex hulls, obtained by rotating the quadrilaterals in the initial list, such that all quadrilaterals in the transformed list are in canonical positions. However, we shall need to rotate hulls so that all of them (quadrilateral, but also triangular and pentagonal) are in canonical positions simultaneously, so we delay this result until we introduce canonical positions for pentagons and triangles.

6.3 Symmetry Breaking for Pentagons

Similarly, the diagonals of the pentagon divide it into eleven different areas (see Fig. 3). For examples, areas 1, 6, and 11 are defined as follows.

```
definition "area_5_1 p q r s t x ⟷
  ¬ ccw p s x ∧ ccw p r x ∧ ccw q t x ∧ ccw q s x ∧ ccw r t x"
definition "area_5_6 p q r s t x ⟷
  ¬ ccw p s x ∧ ¬ ccw p r x ∧ ccw q t x ∧ ccw q s x ∧ ccw r t x"
definition "area_5_11 p q r s t x ⟷
  ¬ ccw p s x ∧ ccw p r x ∧ ¬ ccw q t x ∧ ccw q s x ∧ ccw r t x"
```

All areas from 1 to 5 and all areas from 6 to 10 are mutually symmetric. Therefore, the pentagon can by cyclically permuted so that any point x comes to area 1, area 6, or area 11. We say that a point is in a canonical position if it is inside one of those three areas.

```
definition "area_5_canon p q r s t x ⟷
 ccw p s x ∧ ccw q s x ∧ ccw r t x ∧ (ccw p r x ∨ ccw q t x)"
lemma "area_5_canon p q r s t x ⟷
 area_5_1 p q r s t x ∨ area_5_6 p q r s t x ∨ area_5_11 p q r s t x"
```

The next lemma shows that every pentagon can be rotated to achieve that any point inside it is in a canonical area.

```
lemma
  assumes "in_general_position p, q,r,s,t,x" "convex_polygon [p,q,r,s,t]"
          "ccw p q x" "ccw q r x" "ccw r s x" "ccw s t x" "ccw t p x"
  shows "∃ p'q'r's't'.cyclic_perm [p,q,r,s,t] [p', q', r', s', t'] ∧
                      area_5_canon p' q' r' s' t' x"
```

Again, the previous lemma holds for any point inside the pentagon. We shall rotate every nonempty pentagonal nested convex hull so that the first point of the next (contained) convex hull lies in one of the three pentagon's canonical areas. We shall say that all pentagons in nested convex hulls H are in canonical positions if the following holds.

```
definition nested_hulls_5_canon where
  "nested_hulls_5_canon H ⟷
   (∀ i < length H-1. length (H!i) = 5 ⟶
      area_5_canon (H!i!0) (H!i!1) (H!i!2) (H!i!3) (H!i!4) (H!(i+1)!0))"
```

6.4 Symmetry Breaking for Triangles

The triangle has no diagonals and has just one area, so we must take a different approach for symmetry breaking. If a triangle contains at least two different x and y points then the line xy cuts either pq, or qr, or rp (see Fig. 4).

The three cases are characterized solely by the orientation of the triple x, y, and p, and the triple x, y, and q.

```
definition "cut_3_1 p q r x y ⟷ ¬ ccw x y p ∧ ccw x y q"
definition "cut_3_2 p q r x y ⟷ ¬ ccw x y q ∧ ccw x y r"
definition "cut_3_3 p q r x y ⟷ ¬ ccw x y r ∧ ccw x y p"
```

The points of a triangle can always be cyclically permuted so that for any two points x and y the ray xy cuts the segment pq.

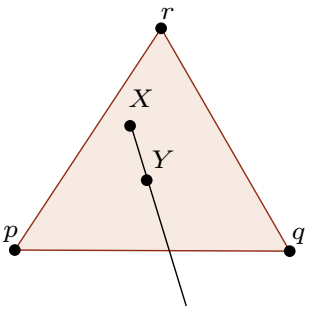


Fig. 4 Triangle in a canonical position

```
lemma
  assumes "in_general_position p, q, r, x, y" "convex_polygon [p, q, r]"
          "ccw p q x" "ccw q r x" "ccw r p x"
  shows "∃ p' q' r'. cyclic_perm [p, q, r] [p', q', r'] ∧
                     cut_3_1 p' q' r' x y"
```

Again, the lemma can be applied to any two points inside the triangle. We shall rotate every triangular nested convex hull that contains at least two points, so that the first two points of the next (contained) convex hull cut the segment formed by the first two points of the triangular hull. We shall say that all triangles in nested convex hulls `H` are in canonical positions if the following holds.

```
definition nested_hulls_3_canon where
  "nested_hulls_3_canon H ⟷
   (∀ i < length H-1. length (H!i) = 3 ∧ length (H!(i+1)) ≥ 2 ⟶
      cut_3_1 (H!i!0) (H!i!1) (H!i!2) (H!(i+1)!0) (H!(i+1)!1))"
```

6.5 The Central Symmetry Result

The central symmetry result claims that nested convex hulls can be cyclically permuted so that all triangles, quadrilaterals and pentagons simultaneously come into canonical positions.

```
lemma
  assumes "nested_hulls H" "in_general_position (set (concat H))"
  shows "∃ H'. nested_hulls H' ∧
               set (concat H') = set (concat H) ∧ length H' = length H ∧
               nested_hulls_3_canon H' ∧
               nested_hulls_4_canon H' ∧
               nested_hulls_5_canon H'"
```

It is proved by induction, starting from inwards and orienting one by one nested convex hull if necessary (using the previous three lemmas that show that rotating a single triangle, quadrilateral or pentagon can bring it to a canonical position with respect to any fixed points contained inside it).

6.6 SAT Encoding of Symmetry Breaking

For a given nested convex hulls structure `Hs` (a list of numbers), we can formulate a SAT formula encoding that all polygons contained in nested convex hulls with the structure `Hs` are in canonical positions. For example, the formula for triangles is the following.

```
definition formula_nested_hulls_3_canon "nat list ⇒ int list list" where
"formula_nested_hulls_3_canon Hs =
  concat (map (λ i.
    if Hs!i = 3 ∧ Hs!(i+1) ≥ 2 then
      let N = listsum Hs; start = listsum (take i Hs)
       in [[-ccw_lit N (start + 3) (start + 4) (start + 0)],
           [ccw_lit N (start + 3) (start + 4) (start + 1)]]
    else [])
  [0..<length Hs - 1])"
```

Abstract characterization of this formula is the following.

```
lemma "val ⊨ (formula_nested_hulls_3_canon Hs) ⟷
  (∀ i < length Hs - 1. Hs!i = 3 ∧ Hs!(i+1) ≥ 2 ⟶
  (let N = listsum Hs; start = listsum (take i Hs)
    in val ⊭ ccw_lit N (start + 3) (start + 4) (start + 0) ∧
       val ⊨ ccw_lit N (start + 3) (start + 4) (start + 1)))"
```

Formulas encoding that all quadrilaterals and pentagons are in canonical positions are formulated and characterized quite similarly, so we omit those definitions from the text (they are available in the Isabelle/HOL proofs).

7 Correctness of the SAT Encoding

With all elements prepared, we can easily build a formula for each possible nested convex hull structure `Hs`, that is then handled to the SAT solver to show its unsatisfiability.

```
definition formula where
  "formula m Hs =
   (formula_ax4 (listsum Hs)) @
   (formula_ax5 (listsum Hs)) @
   (formula_no_polygon m (listsum Hs)) @
   (formula_nested_hulls_nested Hs) @
   (formula_nested_hulls_polygons Hs) @
   (formula_nested_hulls_3_canon Hs) @
   (formula_nested_hulls_4_canon Hs) @
   (formula_nested_hulls_5_canon Hs)"

definition simplified_formula where
  "simplified_formula m H = simplify_formula (formula m Hs)"
```

The simplification turns out to be significant due to the presence of many unit literals. Of course, the SAT solver would itself do this simplification before the solving begins, but it seems that dealing with smaller formulas helps the communication between Isabelle/HOL and underlying SAT solvers.

Abstract characterization of satisfiability of this formula (characterization of `val ⊨ (formula m Hs)`) is easily formed by combining abstract characterization for each component (those have already been proved and described). This helps to prove the following characterization of the final formula.

```
lemma
  assumes "in_general_position S" "card S > 0"
  "m ≥ 3" "¬ (∃ p. set p ⊆ S ∧ length p = m ∧ convex_polygon p)"
  "nested_hulls H" "set (concat H) = S"
  "nested_hulls_3_canon H"
  "nested_hulls_4_canon H"
  "nested_hulls_5_canon H"
  shows "∃ val. val ⊨ (formula m (map length H))"
```

The proof of this lemma proceeds as follows. Nested convex hulls can be concatenated to form a list of points (starting from the first, i.e., the outermost hull). Let `g` be a function that maps indices to points in that list (i.e., `λ i. (concat H)! i`), and let `f` be its inverse mapping points to their indices. If `n` is the cardinality of `S`, functions `f` and `g` are bijections between points in `S` and the set `{0..<n}`. Then the valuation `val` that satisfies the formula can be defined. A variable `v` shall be true in the valuation if it is obtained from indices of three points that are oriented counterclockwise, and false otherwise.

```
"val = (λ v. ∃ p q r. p < q ∧ q < r ∧ r < n ∧
                      v = ccw_var n p q r ∧ ccw (g p) (g q) (g r))
```

Now it is easy to prove (using injectivity of `ccw_var`) that the following holds.

```
"∀ p q r. p < q ∧ q < r ∧ r < n ⟶
          (val (ccw_var n p q r) ⟷ ccw (g p) (g q) (g r))"
"∀ p q r. distinct [p, q, r] ∧ {p, q, r} ⊆ {0..<n} ⟶
          (val ⊨ ccw_lit n p q r ⟷ ccw (g p) (g q) (g r))"
```

Then it is easy to show that this valuation satisfies the formula (using the abstract characterization of the formula, assumptions and axioms). □.

Next we can prove our central theorem about SAT reduction, which connects the unsatisfiability of all SAT formulas for all possible nested convex hull structures with n points containing not m-gon, with the existence of a convex m-gon in each set of n points in a general position.

```
theorem
  assumes "in_general_position S" "finite S" "card S = n" "m ≥ 3"
          "∀ H ∈ set (hull_structure m n).
               ¬ (∃ val. val ⊨ (simplified_formula m H))"
  shows "∃ p. set p ⊆ S ∧ length p = m ∧ convex_polygon p"
```

The proof combines lemmas that we have already shown: (i) existence of nested convex hulls for each set of points in a general position, (ii) the fact that nested convex hulls can be cyclically permuted into canonical position, (iii) the fact that `hull_structure` function computes every possible structure of nested convex hulls that do not contain an m-gon, and (iv) the characterization of the final formula given by the previous lemma. □.

Finally, several cases of the Happy Ending conjecture are proved.

```
theorem
  assumes "in_general_position S" "card S = 9"
  shows "∃ p. set p ⊆ S ∧ length p = 5 ∧ convex_polygon p"
```

The theorem is reduced to the previous one, and the proof requires proving that for any nested convex hull structure, the final formula is unsatisfiable, and that is done by normalization and by applying a SAT solver. More details shall be given in Sect. 8.

Note that the proof is done within the `CCsystem` locale, so the proved cases of the Happy Ending conjecture hold in all CC systems (systems that satisfy the axioms). The $\mathbb{R}^2$ Cartesian plane is an important special case.

8 Experimental Results

We have done all the experiments on a notebook computer with an Intel Core™ i3 2.3 GHz processor and 4 GB RAM memory, running Linux. We have used Isabelle/HOL 2015 system.

Table 1 Experimental results for $m = 5$ and $n = 9$

No.	Hulls structure	Time (normalization + solving + verifying) Isabelle/HOL (s)
1.	[3, 3, 3]	41.9
2.	[3, 4, 2]	38.3
3.	[4, 3, 2]	36.5
4.	[4, 4, 1]	32.6
	Total	149.3

We have shown that the Happy Ending conjecture holds for $(m, n) \in \{(3, 3), (4, 5), (5, 9), (6, 17)\}$. The first three cases are done completely within Isabelle/HOL. We used the tactic `normalization` that performs normalization by evaluation (NBE) [1] (it bypasses the Isabelle kernel, but is significantly faster than simplification) and generates SAT formulas by unfolding and evaluating all the relevant definitions, and then the tactic `sat` that employs MiniSAT solver and formally checks the obtained unsatisfiability proof [26]. The first two cases ($m = 3$ and $m = 4$) are proved almost momentarily (the first by showing that `hull_structure 3 3` is empty), and the time for proving the third case ($m = 5$) is reported in Table 1. Note that, due to parallel proof-checking, these times do not add up, and the wall-clock time required to check the proof of the whole theorem is a bit less (it is around 107 s).

In the last case ($m = 6$), the formulas were exported to DIMACS[2] files (using a simple ML script) and the MiniSAT solver was externally applied (bypassing the normalization, and proof verification steps that proved to be too time and memory consuming in this case). To achieve higher reliability, certified SAT solving was applied and a patched[3] version of MiniSAT 2.2, that generates unsatisfiability proofs in DRAT format, was used. The solver was used with the default options, but without formula preprocessing (the `core` instead of the `simp` mode was used, as the `simp` mode turns out to be less efficient). Proof generation during solving did not incur a significant overhead (when compared with the original MiniSAT, the patched version was only about 2% slower). The generated proofs were converted to GRAT format (by an unverified GRATgen tool) and then checked using the formally verified GRATchk tool [15]. The SAT solving and proof-checking time were almost the same (around 1700 s vs. around 1600 s total). In earlier experiments we checked the proofs by the (unverified) DRAT-trim tool [29] and the proof-checking time was even a bit higher (around 1650 s total) then with the GRAT toolchain (GRATgen and GRATchk) that offers much stronger formal correctness guarantees.

Generating and exporting DIMACS formulas from Isabelle/HOL was the most time consuming phase (it took more time than solving the formulas by MiniSAT and verifying their unsatisfiability proofs by DRAT-trim or by the GRAT tool-chain). Isabelle/HOL definitions could have been optimized. For example, instead of simplifying formulas after they are generated, the simplification could have been done during the formula generation. Also, formulas for different nested hull structures share a common part that depends only on m and n (it consists of `formula_ax4`, `formula_ax5`, and `formula_no_polygon`), and when it is cached when computed for the first time, the total time for formula generation drops from

[2] This is the standard input format for SAT solvers.

[3] The patch was taken from the SAT 2014 competition website http://www.satcompetition.org/2014/description.shtml.

more than 7000 s to around 2800 s. However, we did not push the Isabelle/HOL formula generation to its limits, but instead we have implemented a simple (but unverified) C++ tool that generates DIMACS files. The generated formulas are exactly the same as the ones generated from Isabelle/HOL, but the time to generate them is an order of magnitude smaller.

All times are reported in Table 2.

Some other results In their pen-and-paper attempt, Dehnhardt et al. have proved a lemma claiming that any set of points in a general position where the outer hull is a pentagon that contains a triangle and at least five points inside the triangle contains a convex hexagon [7]. It was easy to formally verify this result within our framework. We have examined the hull structures [5, 3, 3, 2], [5, 3, 4, 1], and [5, 3, 5] and proved that all of them contain a convex hexagon. Note that this result is somewhat surprising as there are only 13 points in these configurations, while in general 17 points are necessary to force a convex hexagon. The smaller number of points makes the formula smaller and much simpler to generate and prove unsatisfiable.

Such partial results can be used to speed up the central proof. For example, the case [5, 3, 5, 4] need not be verified since it is subsumed by the case [5, 3, 5]. However, it turns out that proving unsatisfiability for the case [5, 3, 5, 4] is very easy (as Table 2 shows, it takes only about 2 s). Our explanation is simple: the SAT solver easily finds the unsatisfiable core corresponding to the embedded [5, 3, 5] configuration. Results in Table 2 indicate that this is not a coincidence — all the cases that could be eliminated are solved very quickly. Therefore, there would be not much gain in the total solving time if those cases were eliminated because they are subsumed by some earlier results. However, the time to generate those formulas can be saved and in some settings (e.g., when formulas are generated by Isabelle/HOL) that time is not negligible. Still, we did not formalize this technique.

Other nested convex hull structures (not subsumed by some smaller nested convex hull structures) that have less than 17 points and force a convex hexagon are the following: [5, 4, 5], [3, 5, 3, 4], [5, 4, 3, 3], [5, 4, 4, 2], and [4, 5, 3, 4]. Note that [5, 4, 5, 1] also forces a convex hexagon (since [5, 4, 5] does), so every configuration with at least six points in a quadrilateral inside a pentagon also forces a convex hexagon.

Note that in the case $m = 5$ and $n = 9$, the configurations with the nested convex hulls of the structure [4, 3, 1] force a convex pentagon (again, note that this is just 8 points, and not 9, as it is required in the general case).

9 Further Work

We did not yet investigate the case $m = 7$ and $n = 33$. The approach used for $m \leq 6$ is not directly applicable to the case $m = 7$, due to the size of generated SAT formulas (although the increase from $m = 6$ to $m = 7$ is small, the increase from $n = 17$ to $n = 33$ is really huge). The main problem comes from the `formula_no_polygon` formula that would contain 3 billion clauses with more than 100 billion literals (before simplification). Therefore, the eager approach to generate all the clauses a priori is not possible. One possibility would be to try a lazy approach (e.g., to define an SMT theory that would be devoted to handling the `no_polygon` constraint). The case of $m = 7$ would also require to introduce some symmetry breaking for hexagons, and probably to further improve symmetry breaking for pentagons, quadrilaterals, and triangles. Even with all those modifications it is hard to estimate whether it would be possible to prove the conjecture, or even some of its simpler cases. Note that for $m = 6$ there are 73 possible hull structures, while for $m = 7$, that number rises to

Table 2 Experimental results for $m = 6$ and $n = 17$

No.	Hulls structure	Generating		Solving	Verifying
		Isabelle/HOL (s)	C++ (s)	MiniSAT 2.2 (s)	GRAT (gen + chk) (s)
1.	[3, 3, 3, 3, 3, 2]	122.2	2.92	3.94	4.47 + 2.52
2.	[3, 3, 3, 3, 4, 1]	119.5	2.79	7.60	7.73 + 2.14
3.	[3, 3, 3, 3, 5]	100.1	3.01	6.61	6.68 + 2.37
4.	[3, 3, 3, 4, 3, 1]	116.4	2.91	23.09	22.74 + 6.26
5.	[3, 3, 3, 4, 4]	97.7	2.90	29.51	30.65 + 7.52
6.	[3, 3, 3, 5, 3]	94.4	2.86	25.73	28.87 + 8.53
7.	[3, 3, 4, 3, 3, 1]	104.6	2.90	21.62	20.10 + 4.62
8.	[3, 3, 4, 3, 4]	90.6	2.89	22.58	22.70 + 11.48
9.	[3, 3, 4, 4, 3]	106.2	2.92	83.41	77.46 + 27.93
10.	[3, 3, 4, 5, 2]	91.6	2.82	46.08	42.17 + 13.35
11.	[3, 3, 5, 3, 3]	100.1	2.87	19.62	14.69 + 3.61
12.	[3, 3, 5, 4, 2]	89.7	2.86	33.05	25.93 + 4.28
13.	[3, 3, 5, 5, 1]	92.0	2.72	25.75	15.84 + 3.57
14.	[3, 4, 3, 3, 3, 1]	103.0	2.89	8.04	7.59 + 2.94
15.	[3, 4, 3, 3, 4]	109.1	2.89	14.56	11.32 + 3.62
16.	[3, 4, 3, 4, 3]	135.0	2.92	37.21	30.34 + 5.84
17.	[3, 4, 3, 5, 2]	92.2	2.81	21.91	14.82 + 3.28
18.	[3, 4, 4, 3, 3]	82.2	2.93	43.90	27.51 + 4.80
19.	[3, 4, 4, 4, 2]	91.4	2.91	69.86	47.98 + 6.64
20.	[3, 4, 4, 5, 1]	97.7	2.76	44.10	31.24 + 5.32
21.	[3, 4, 5, 3, 2]	85.0	2.85	36.53	21.91 + 4.30
22.	[3, 4, 5, 4, 1]	89.3	2.80	42.70	27.30 + 4.46
23.	[3, 4, 5, 5]	105.6	2.87	55.39	34.96 + 5.10
24.	[3, 5, 3, 3, 3]	87.2	2.88	0.94	2.46 + 2.14
25.	[3, 5, 3, 4, 2]	82.1	2.95	1.00	2.44 + 2.03
26.	[3, 5, 3, 5, 1]	106.8	2.70	0.84	2.37 + 2.14
27.	[3, 5, 4, 3, 2]	93.6	2.86	9.64	6.46 + 2.43
28.	[3, 5, 4, 4, 1]	84.7	2.82	6.70	5.43 + 2.36
29.	[3, 5, 4, 5]	90.1	2.87	2.11	4.44 + 2.06
30.	[3, 5, 5, 3, 1]	95.6	2.85	23.71	12.05 + 3.53
31.	[3, 5, 5, 4]	104.1	2.87	23.55	11.26 + 3.48
32.	[4, 3, 3, 3, 3, 1]	95.7	2.89	9.00	6.45 + 2.27
33.	[4, 3, 3, 3, 4]	100.8	2.89	9.69	6.93 + 3.34
34.	[4, 3, 3, 4, 3]	83.5	2.92	19.33	14.53 + 3.26
35.	[4, 3, 3, 5, 2]	87.6	2.80	8.94	6.78 + 2.99
36.	[4, 3, 4, 3, 3]	119.6	2.92	30.89	21.10 + 4.12
37.	[4, 3, 4, 4, 2]	92.1	2.91	47.10	34.42 + 5.04

Table 2 continued

No.	Hulls structure	Generating		Solving	Verifying
		Isabelle/HOL (s)	C++ (s)	MiniSAT 2.2 (s)	GRAT (gen + chk) (s)
38.	[4, 3, 4, 5, 1]	82.5	2.75	34.44	24.65 + 4.00
39.	[4, 3, 5, 3, 2]	91.9	2.83	27.96	20.77 + 3.25
40.	[4, 3, 5, 4, 1]	108.6	2.81	35.22	27.51 + 3.88
41.	[4, 3, 5, 5]	80.8	2.80	37.94	29.19 + 4.16
42.	[4, 4, 3, 3, 3]	100.8	2.92	16.42	9.64 + 2.64
43.	[4, 4, 3, 4, 2]	86.6	2.92	30.02	20.66 + 3.40
44.	[4, 4, 3, 5, 1]	80.5	2.75	24.35	17.59 + 3.01
45.	[4, 4, 4, 3, 2]	92.5	2.89	65.35	39.58 + 4.72
46.	[4, 4, 4, 4, 1]	95.8	2.92	73.74	52.27 + 5.87
47.	[4, 4, 4, 5]	82.6	2.84	75.48	48.83 + 10.26
48.	[4, 4, 5, 3, 1]	88.6	2.88	69.12	45.30 + 12.57
49.	[4, 4, 5, 4]	90.2	2.89	83.75	59.86 + 7.50
50.	[4, 5, 3, 3, 2]	109.9	2.84	0.78	2.39 + 5.57
51.	[4, 5, 3, 4, 1]	149.5	2.78	0.82	2.45 + 3.91
52.	[4, 5, 3, 5]	81.6	2.80	0.76	2.25 + 1.93
53.	[4, 5, 4, 3, 1]	91.8	2.89	49.85	30.71 + 4.97
54.	[4, 5, 4, 4]	92.3	2.90	45.19	24.82 + 4.18
55.	[4, 5, 5, 3]	83.5	2.85	46.47	27.72 + 4.23
56.	[5, 3, 3, 3, 3]	86.2	2.87	2.25	2.67 + 2.05
57.	[5, 3, 3, 4, 2]	96.6	2.85	2.48	3.02 + 2.82
58.	[5, 3, 3, 5, 1]	106.7	2.71	1.65	2.39 + 2.00
59.	[5, 3, 4, 3, 2]	78.8	2.85	1.96	2.46 + 2.33
60.	[5, 3, 4, 4, 1]	115.4	2.81	1.65	2.31 + 1.94
61.	[5, 3, 4, 5]	91.2	2.82	1.83	2.22 + 2.01
62.	[5, 3, 5, 3, 1]	84.1	2.84	1.53	2.47 + 2.29
63.	[5, 3, 5, 4]	91.6	2.85	1.59	2.39 + 2.01
64.	[5, 4, 3, 3, 2]	110.6	2.84	1.45	2.20 + 1.95
65.	[5, 4, 3, 4, 1]	80.5	2.80	2.44	2.82 + 2.23
66.	[5, 4, 3, 5]	98.3	2.80	1.43	2.02 + 2.04
67.	[5, 4, 4, 3, 1]	106.2	2.89	7.24	5.37 + 3.52
68.	[5, 4, 4, 4]	92.4	2.94	6.60	5.35 + 2.98
69.	[5, 4, 5, 3]	83.6	2.87	5.52	4.89 + 2.71
70.	[5, 5, 3, 3, 1]	96.5	2.85	4.99	4.37 + 2.38
71.	[5, 5, 3, 4]	94.3	2.92	5.29	4.63 + 6.02
72.	[5, 5, 4, 3]	86.8	2.89	11.46	7.44 + 8.15
73.	[5, 5, 5, 2]	87.8	2.80	8.79	6.50 + 7.47
	Total	7016.7	208.53	1704.05	1263.82 + 332.6

22765. Since in the case $m = 6$ there is not too much variation in solving time between different hull structures, we could expect that if several cases for $m = 7$ turn out to be provable, then the whole theorem can be attacked by using a large cluster computer and parallelization (formulas for different nested convex hull structures can be generated and checked by different processors).

An interesting starting point that could be examined is what is the minimal number of points contained in a triangle inside of a hexagon, so that a convex heptagon is guaranteed. Namely, as we already noted, it takes only one point in a triangle inside a quadrilateral to guarantee a convex pentagon (this is only 8 instead of 9 points generally needed for a convex pentagon) and it takes only five points in a triangle inside a pentagon to guarantee a convex hexagon (this is only 13 instead of 17 points needed for a convex hexagon). Therefore, one can expect that significantly less points than 33 are needed to guarantee a convex heptagon in configurations with points in a triangle inside a hexagon.

Further research might also try to investigate symmetry breaking more thoroughly so that it could be guaranteed that each potential model of the formula is essentially unique. This would additionally speed up the SAT solving, but could also be used to bring some more explanations into the proofs by helping us to identify essentially non-equivalent configurations of points.

We also plan to investigate if it would be possible to replace SAT solvers by some other off-the-shelf automated reasoning tools (e.g., by first-order resolution provers).

10 Conclusions

We have proved that Erdős–Szekeres "Happy Ending" conjecture holds for triangles ($m = 3, n = 3$), quadrilaterals ($m = 4, n = 5$), pentagons ($m = 5, n = 9$) and hexagons ($m = 6, n = 17$). Although all these results were previously known, we have improved the proofs in several important directions.

The previous proof for hexagons ($m = 6, n = 17$) required tremendous computing power (Szekeres and Peters report that it took more than 3000 GHz hours to complete), and was done using a cluster computer. In its least reliable setting (when only unverified C++ script and MiniSAT are used), our proof can be completed on an average PC computer in just half an hour, i.e., it requires only a bit more than 1 GHz hour. The time increases in a more reliable setting (when Isabelle/HOL, MiniSAT and GRAT toolchain are used), but still takes only around 6–7 GHz hours to complete, and can quickly be completed in a couple of hours on an average PC.

Our proof is significantly more reliable than the proof of Szekeres and Peters. All the theory behind our proof is formalized within Isabelle/HOL. The cases of $3 \leq m \leq 5$ are fully formally verified within Isabelle/HOL (except that NBE is used which bypasses the kernel). The case $m = 6$ is not done completely within Isabelle/HOL, but the proof is still quite reliable, even when the unverified C++ script is used. Namely, the C++ script generates the DIMACS formula by strictly following the specification of that formula which is verified within Isabelle/HOL. The C++ code is simple and straightforward, and there is very little chance that something goes wrong (indeed, comparing files generated by the C++ script and those exported from Isabelle/HOL confirms that they are exactly the same). There is much more chance that an error is made in a background mathematical reasoning, but this is eliminated by mechanically verifying the background theory in Isabelle/HOL. Comparing to the C++ script, the reliability is increased if formulas are generated and exported directly from

Isabelle/HOL (ML script that does the export is rather trivial), but at a price of consuming more time.

Potential errors made by the SAT solver are excluded by checking the unsatisfiability proofs generated by the solver (within Isabelle for $3 \leq m \leq 5$, and by using the GRAT toolchain for $m = 6$). Although Isabelle/HOL offers SAT proof-checking, the state-of-the-art technology (the DRAT-trim approach) has not yet been integrated into the system, so the $m = 6$ case could not be verified fully within Isabelle/HOL. However, checking the proof by the GRATchk tool gives us very high reliability, since GRATchk is formally verified within Isabelle/HOL.

Unlike the Szekeres and Peters proof that treats the pentagon case ($m = 5$) and the hexagon case ($m = 6$) somewhat differently, our approach treats all cases ($3 \leq m \leq 6$) uniformly, and the same code is used to prove all cases (including the additional results discussed in Sect. 8).

Similar to Szekeres and Peters proof, our proof is also amenable to parallelization. Namely formulas corresponding to different nested convex hull structures can be parallely generated and solved.

A byproduct of our formalization is the formalization of the theory of CC systems [14], and computational geometry notions and algorithms (e.g., the incremental algorithm for convex hulls) based on it, which can be useful on its own. Although this has been formalized in Coq [22], up to our best knowledge, this is the first such formalization in Isabelle/HOL.

Lessons learned from this effort is that it often pays off to use well developed and established reasoning tools, rather than implementing proving procedures from scratch. Namely, Szekeres and Peters [25] implemented their naive backtracking search procedure from scratch, and it turns out that a very significant speedup can be achieved by simply replacing that procedure by a modern SAT solver (our preliminary experiments have estimated that only by applying SAT solvers to Peters and Szekeres approach, one could reduce the proving time from several thousands to several tens of hours).

However, the quality of the results that we have obtained was not possible only due to SAT solving, but also due to change in the problem representation that has enabled us to split a large problem into many simpler ones and to apply symmetry breaking. The main idea of examining the structure of nested convex hulls is taken from pen-and-paper proofs [5,7], showing that it often pays of to examine smart pen-and-paper approaches and imitate them by automated reasoning.

Pen-and-paper proofs often offer more explanation than those automatically generated by the machine. However, illustrative lemmas can often provide enough details to understand why the theorem holds. For example, a subtle and important insight in the pen-and-paper proof by Dehnhardt et al. [7] was that five points inside a triangle inside a pentagon guarantee a convex hexagon, and that same fact was easily shown within automated proof. On the other hand, the fact that the number of different possible nested convex hull structures i.e., the list of cases that need to be analyzed in the proof was wrongly identified in two different publications, indicate the inherent unreliability of all pen-and-paper proofs (missing cases are often problematic).

Symmetry breaking has proved to be extremely valuable technique in reducing the problem complexity, and it pays off to seek problem representations that allow symmetry breaking.

Finally, proofs-by-computation require often complex algorithms, and the only way to fully trust them is to implement and verify all procedures within modern proof assistants. We estimate that the additional formalization effort took around two full research weeks (formal proof documents consume around 5000 loc), but we are convinced that in computer-based theorem proving machine checked proofs should become the norm.

References

1. Aehlig, K., Haftmann, F., Nipkow, T.: A compiled implementation of normalization by evaluation. In: Mohamed, O.A., Munoz, C., Tahar, S. (eds.) Theorem Proving in Higher Order Logics (TPHOLs 2008), LNCS, vol. 5170, pp. 39–54. Springer, Berlin (2008)
2. Avigad, J., Harrison, J.: Formally verified mathematics. Commun. ACM **57**(4), 66–75 (2014)
3. Ballarin, C.: Interpretation of locales in Isabelle: theories and proof contexts. In: Proceedings of Mathematical Knowledge Management, MKM, pp. 31–43 (2006)
4. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability. IOS Press, Amsterdam (2009)
5. Bonnice, W.E.: On convex polygons determined by a finite planar set. Am. Math. Mon. **81**, 749752 (1974)
6. Cruz-Filipe, L., Marques-Silva, J., Schneider-Kamp, P.: Efficient certified resolution proof checking. In: Proceedings of Automated Deduction—CADE-26—26th International Conference on Automated Deduction, Gothenburg, Sweden, LNCS. Springer (2017)
7. Dehnhardt, K., Harborth, H., Längi, Z.: A partial proof of the Erdős–Szekeres conjecture for hexagons. J. Pure Appl. Math. Adv. Appl. **2**, 6986 (2009)
8. Erdős, P., Szekeres, G.: A combinatorial problem in geometry. Compos. Math. **2**, 463–470 (1935)
9. Hales, T.C. (ed.): Notices of the AMS: Special Issue on Formal Proof, vol. 55(11). American Mathematical Society (2008)
10. Harrison, J.: HOL light: a tutorial introduction. In: Proceedings of Formal Methods in Computer-Aided Design, First International Conference, FMCAD'96, Palo Alto, California, USA, pp. 265–269 (1996)
11. Hölldobler, S., Manthey, N., Philipp, T., Steinke, P.: Generic CDCL—a formalization of modern propositional satisfiability solvers. In: POS@ SAT, pp. 89–102 (2014)
12. Huet, G., Herbelin, H.: 30 years of research and development around Coq. In: Principles of Programming Languages, POPL, pp. 249–250 (2014)
13. Kalbfleisch, J.D., Kalbfleisch, J.G., Stanton, R.G.: A combinatorial problem on convex n-gons. In: Proceedings of Louisiana Conference on Combinational Graph Theory Computing, Louisiana State University, Baton Rouge (1970)
14. Knuth, D.E.: Axioms and Hulls, LNCS, vol. 606. Springer, Berlin (1992)
15. Lammich, P.: Efficient verified (un)sat certificate checking. In: Proceedings of Automated Deduction—CADE-26—26th International Conference on Automated Deduction, Gothenburg, Sweden, LNCS. Springer (2017)
16. Marić, F.: Formalization and implementation of modern SAT solvers. J. Autom. Reason. **43**(1), 81–119 (2009)
17. Marić, F.: Formal verification of a modern SAT solver by shallow embedding into Isabelle/HOL. Theor. Comput. Sci. **411**(50), 4333–4356 (2010)
18. Marić, F.: A survey of interactive theorem proving. Zb. Rad. **18**, 173–223 (2015)
19. Morris, W., Soltan, V.: The Erdős–Szekeres problem on points in convex position—a survey. Bull. Am. Math. Soc. **37**, 437–458 (2000)
20. Morris, W., Soltan, V.: The Erdős–Szekeres Problem. Springer, Cham (2016)
21. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL—A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
22. Pichardie, D., Bertot, Y.: Formalizing convex hull algorithms. In: Boulton, R.J., Jackson, P.B. (eds.) Proceedings of Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001 Edinburgh, Scotland, UK, pp. 346–361. Springer, Berlin (2001)
23. Shankar, N., Vaucher, M.: The mechanical verification of a DPLL-based satisfiability solver. Electron. Notes Theor. Comput. Sci. **269**, 3–17 (2011)
24. Suk, A.: On the Erdős-Szekeres convex polygon problem. J. Am. Math. Soc. **30**, 1047–1053 (2017)
25. Szekeres, G., Peters, L.: Computer solution to the 17-point Erdős–Szekeres problem. ANZIAM J. **48**(2), 151–164 (2006)
26. Weber, T.: Efficiently checking propositional resolution proofs in Isabelle/HOL. In: Benzmüller, C., Fischer, B., Sutcliffe, G. (eds.) Proceedings of the 6th International Workshop on the Implementation of Logics, CEUR Workshop Proceedings, vol. 212, pp. 44–62 (2006)
27. Weber, T.: Integrating a SAT solver with an LCF-style theorem prover. Electr. Notes Theor. Comput. Sci. **144**(2), 67–78 (2006)
28. Wenzel, M.: Isabelle/Isar—a generic framework for human-readable proof documents. In: Matuszewski, R., Zalewska, A. (eds.) From Insight to Proof—Festschrift in Honour of Andrzej Trybulec, Studies in Logic, Grammar, and Rhetoric, vol. 10(23). University of Bialystok (2007)
29. Wetzler, N., Heule, M.J.H., Hunt, W.A.: Drat-trim: Efficient checking and trimming using expressive clausal proofs. In: Sinz, C., Egly, U. (eds.) Theory and Applications of Satisfiability Testing—SAT 2014:

17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, Proceedings, pp. 422–429. Springer, Cham (2014)
30. Wetzler, N.D., et al.: Efficient, mechanically-verified validation of satisfiability solvers. Ph.D. thesis, University of Texas, Austin, USA (2015)