Weak Alternating Automata in Isabelle/HOL

Stephan Merz

Institut für Informatik, Universität München
merz@informatik.uni-muenchen.de

Abstract. Thomas has presented a novel proof of the closure of ω-regular languages under complementation, using weak alternating automata. This note describes a formalization of this proof in the theorem prover Isabelle/HOL. As an application we have developed a certified translation procedure for PTL formulas to weak alternating automata inside the theorem prover.

1 Introduction

The close relationship between ω-automata and temporal logic [10, 8] is one of the cornerstones of the theory that underlies model checking. Traditionally, ω-regular languages have been defined via Büchi automata [1, 7, 8], a straightforward extension of standard finite automata, but operating on infinite words. One of the fundamental results about ω-regular languages establishes their closure under complementation. It was first proven non-constructively by Büchi [1], relying on Ramsey's theorem. Over a period of 25 years, the result has been reproved several times, using variants of Büchi automata to obtain effective constructions, culminating in a paper by Safra [5] that gives an essentially optimal, exponential construction.

An alternative definition of ω-regular languages due to Muller, Saoudi, and Schupp [3] is based on (weak) alternating automata, for which several states can be simultaneously active during a run over a given word. In this framework, complementation can simply be achieved by dualizing the transition relation of the original automaton, avoiding the exponential blowup of Büchi automata. Although the construction is simple, its correctness proof is far from obvious. Thomas [9] has recently given a beautiful presentation of this proof in terms of winning strategies for the class of games associated with weak alternating automata, isolating the complexity of the proof in three independent subproblems. We have formalized Thomas' proof in the interactive theorem prover Isabelle/HOL and believe that this formalization constitutes an interesting case study in formalizing mathematics because it involves fairly advanced mathematical concepts such as logical games and strategies, offering a mix of combinatorial reasoning and of linear and modulus arithmetic that exercise the power of automated proof strategies.

As an application, we have then verified a simple, linear, and compositional construction [11, 2] that associates a weak alternating automaton $\mathcal{A}_\varphi$ with a given formula φ of propositional temporal logic. Because this construction is defined as a set of recursive functions, we may use Isabelle's rewriting machinery to actually evaluate these functions and in this way obtain a certified translation of LTL formulas to automata within the theorem prover.

2 Isabelle/HOL

Isabelle is a generic interactive theorem prover which can be instantiated with different object logics. One popular and well-developed instance has been developed for higher-order logics, based on Church's version of Higher Order Logic. From now on, Isabelle means Isabelle/HOL. Extensive documentation can be found on the Web at `http://isabelle.in.tum.de`; we only highlight some of the syntax required for understanding the remainder of the paper.

We use types constructed by function application ($\Rightarrow$), products ($*$) or records (i.e., tuples with named components). Isabelle also supports inductive definitions of data types.

The syntax for formulas is standard. Isabelle distinguishes between object-level ($\Rightarrow$) and meta-level ($\Longrightarrow$) implication, and similarly for universal quantification, but that distinction is unimportant for our purposes. The notation $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$ is short-hand for the nested meta-level implication $A_1 \Longrightarrow \ldots A_n \Longrightarrow A$.

Definitions of types and operators are collected in theories. Non-recursive operators are defined via the meta-level equality ($\equiv$), recursive operators can be introduced by `primrec` and `recdef` constructs.

We do not present any proofs, because Isabelle proof scripts are not intelligible for human readers, but we usually indicate their complexity in terms of how many interactions were necessary. Besides low-level proof commands such as resolution and instantiation, Isabelle provides higher-level search procedures (tactics) based on rewriting and the classical reasoner, which implements a tableau-based prover for predicate logic and sets. These automatic tactics must be supplied with information about which definitions to expand and which lemmas to use, which requires some expertise and experimentation.

3 Automata, Games, and Strategies

We describe the concepts of the theory of weak alternating automata and their associated games that are used in Thomas' proof as well as their formalization in Isabelle. The Isabelle definitions have been copied verbatim from the input files, except for some pretty-printing to improve readability.

3.1 Positive Boolean Formulas

The transition relation of an alternating automaton is conveniently defined via positive Boolean combinations of its states, represented by the following inductive data type parameterized by type σ.

```
datatype σ pboolean =
    Atomic σ
  | And    (σ pboolean) (σ pboolean)
  | Or     (σ pboolean) (σ pboolean)
```

Straightforward recursive functions compute the set of atoms that occur in a positive Boolean formula, and its models.

```
atoms :: σ pboolean ⇒ σ set
atoms (Atomic s) = {s}
atoms (And p q)  = (atoms p) ∪ (atoms q)
atoms (Or f g)   = (atoms f) ∪ (atoms g)

models :: σ pboolean ⇒ σ set set
models (Atomic s) = { M | s ∈ M }
models (And p q)  = (models p) ∩ (models q)
models (Or p q)   = (models p) ∪ (models q)
```

We will mainly be interested in "small" models that are subsets of the atoms contained in a formula.

```
smodels :: σ pboolean ⇒ σ set set
smodels p ≡ (models p) ∩ 𝒫(atoms p)
```

The *dual formula* $\tilde{p}$ of a positive Boolean formula p is obtained by exchanging conjunctions and disjunctions.

```
dual_form (Atomic s) = Atomic s
dual_form (And p q)  = Or (dual_form p) (dual_form q)
dual_form (Or p q)   = And (dual_form p) (dual_form q)
```

We can prove a number of preliminary lemmas about positive Boolean formulas and their models. For example, every formula has a model, and small models are finite sets. A set M is a model of $\tilde{p}$ iff it contains an element of every model of p

```
M ∈ models (dual_form p) ≡ ∀R ∈ models p. ∃s. s ∈ M ∩ R
```

and a similar relation holds for the small models of p and $\tilde{p}$ (Thomas considers minimal models, his "Remark 3" asserts a similar relationship). All of these lemmas are proved by induction, followed by invocations of Isabelle's automated tactics.

3.2 Automata and Runs

A weak alternating automaton $\mathcal{A} = (S, s_0, \delta, \rho)$ over alphabet B is given by a set S of states, an initial state $s_0 \in S$, a transition function δ that associates a positive Boolean formula $\delta(s, b)$ with every state $s \in S$ and input symbol $b \in B$, and a ranking function $\rho : S \to \mathbb{N}$ such that $\rho(t) \leq \rho(s)$ whenever t occurs in some formula $\delta(s, b)$ (this restriction is what makes $\mathcal{A}$ "weak"). We represent automata via a record type (σ, β) `waa` and a well-formedness predicate `is_waa`.

```
record (σ,β) waa =
  states    :: σ set
  initial   :: σ
  trans     :: [σ, β] ⇒ σ pboolean
  rank      :: σ ⇒ nat

is_waa :: (σ,β) waa ⇒ bool
is_waa auto ≡ initial auto ∈ states auto
            ∧ ∀ s ∈ states auto. ∀b.
                    atoms(trans auto s b) ⊆ states auto
                  ∧ ∀ t ∈ atoms(trans auto s b).
                           rank auto t ≤ rank auto s
```

Runs of alternating automata over ω-words are often represented as infinite trees of states where the successors of a state s and input symbol b are given by some model of $\delta(s, b)$. Thomas suggests instead to represent runs as infinite dags, which can be formalized as a record that contains the root state and a function that returns the set of successor states for state s at a given depth in the dag. We also define two type synonyms that will be used to represent ω-words and other infinite sequences.

```
record σ dag =
  root     :: σ
  succs    :: [nat, σ] ⇒ σ set
types
  β word = nat ⇒ β
  σ seq  = nat ⇒ σ
```

The following definitions introduce the set of run dags of an automaton $\mathcal{A}$ over a given word w and the set of paths through an infinite dag:

```
run_dags :: [(σ,β) waa, β word] ⇒ σ dag set
run_dags auto w ≡
  {dg | root dg = initial auto
      ∧ ∀i s. succs dg i s ∈ smodels (trans auto s (w i))}
paths :: σ dag ⇒ (σ seq) set
paths dg ≡ {pi | pi 0 = root dg
               ∧ ∀i. pi (i+1) ∈ succs dg i (pi i)}
```

It remains to define the acceptance condition. Because ranks are decreasing along any path in a run dag, they must eventually stabilize. Now, the run of a weak alternating automaton is accepting iff for every path through the dag, this "limit rank" is even. Equivalently, the least rank assumed along any path through the dag must be even.[1]

```
ranks :: [σ seq, σ ⇒ nat] ⇒ nat set
```

[1] $\circ$ denotes function composition; the range of a function `f` is denoted by `range f`.

```
ranks pi f ≡ range (f ∘ pi)

least_rank :: [σ seq, σ ⇒ nat] ⇒ nat
least_rank pi f ≡ LEAST i. i ∈ ranks pi f

acc_path  :: [σ seq, (σ,β) waa] ⇒ bool
acc_path pi auto ≡ (least_rank pi auto) mod 2 = 0

is_accepting :: [(σ,β) waa, σ dag] ⇒ bool
is_accepting auto dg ≡ ∀pi ∈ paths dg. acc_path pi auto
```

Finally, the language defined by an automaton is the set of words for which there exists some accepting run dag.

```
language :: (σ,β) waa ⇒ β word set
language auto ≡ {w | ∃dg ∈ run_dags auto w. is_accepting auto dg}
```

3.3 Games and Strategies

Logical games have become popular tools in semantics as well as in automata theory. The word problem for a weak alternating automaton $\mathcal{A}$ and an ω-word w can be visualized as a two-person game between A(utomaton) and P(athfinder) where A tries to demonstrate the existence of an accepting run, while P tries to spoil A's efforts. Every draw in the game consists of a move by A, followed by a move of P. For his i'th move, player A sees a state s of the automaton and chooses some (small) model M of $\delta(s, w_i)$, exploiting the non-determinism of automaton $\mathcal{A}$. Player P, trying to find some path in the run whose minimum rank is odd, then chooses some state $t \in M$ for the next round of the game. The outcome of the game is a path through a run dag of $\mathcal{A}$ for input w.

We start by defining type synonyms for the positions of the two players. A game is represented as an ω-sequence of pairs of positions.

```
types
  σ Apos = σ
  σ Ppos = σ set
  σ play = (σ Apos * σ Ppos) seq
```

These definitions differ from those used by Thomas in that he defines positions as pairs (i, s) resp. (i, M) where i gives the index of the current round. Doing so would pollute our terms with projection functions, so we provide i as an extra parameter whenever necessary. The following functions define the set of legal moves of either player at any position, as well as the initial position.

```
Amoves :: [(σ,β) waa, β word, nat, σ Apos] ⇒ σ Ppos set
Amoves auto w i s ≡ smodels (trans auto s (w i))

Pmoves :: [(σ,β) waa, β word, nat, σ Ppos] ⇒ σ Apos set
Pmoves auto w i M ≡ M
```

```
init_pos :: (σ,β) waa ⇒ σ Apos
init_pos auto ≡ initial auto
```

A draw sequence is an ω-sequence of alternating positions that correspond to legal moves; a play is a draw sequence that starts at the initial position. The outcome of a play is its projection on the positions of player A, i.e. on the states of the automaton.[2]

```
drawseqs :: [(σ,β) waa, β word, σ Apos] ⇒ σ play set
drawseqs auto w s ≡
  { pl | fst (pl 0) = s
      ∧ ∀i. snd (pl i) ∈ Amoves auto w i (fst (pl i))
      ∧ ∀i. fst (pl (i+1)) ∈ Pmoves auto w i (snd (pl i))}

plays :: [(σ,β) waa, β word] ⇒ σ play set
plays auto w ≡ drawseqs auto w (init_pos auto)

outcome :: σ play ⇒ σ seq
outcome pl ≡ fst ∘ pl

outcomes :: [(σ,β) waa, β word] ⇒ σ seq set
outcomes auto w ≡  outcome `` (plays auto w)
```

Player A wins if the least rank among the states in the outcome of the play is odd, otherwise P wins.

```
Awins :: [(σ,β) waa, σ play] ⇒ bool
Awins auto pl ≡ acc_path (outcome pl) auto

Pwins :: [(σ,β) waa, σ play] ⇒ bool
Pwins auto pl ≡ ¬ (Awins auto pl)
```

The question of interest is whether either player can force a win in the play for the given automaton and input word by following a strategy. It turns out that for weak alternating automata, it suffices to consider local (i.e., memoryless) strategies, where the next move is determined from the current position alone. We introduce types and well-formedness predicates for strategies to Isabelle.[3]

```
types
  σ Astrat = [nat, σ Apos] ⇒ σ Ppos
  σ Pstrat = [nat, σ Ppos] ⇒ σ Apos

isAstrat :: [σ Astrat, (σ,β) waa, β word] ⇒ bool
```

[2] `fst` and `snd` denote the projection functions. The expression `f``S` denotes the image of set `S` under `f`.

[3] These definitions will be revised in section 5.

```
isAstrat strat auto w ≡ ∀i s. strat i s ∈ Amoves auto w i s

isPstrat :: [σ Pstrat, (σ,β) waa, β word] ⇒ bool
isPstrat strat auto w ≡ ∀i M. strat i M ∈ Pmoves auto w i M
```

A strategy is a winning strategy for either player if it guarantees a win provided the player adheres to the strategy. We give the definitions for player A; those for player P are similar.

```
Aadheres :: [σ play, σ Astrat] ⇒ bool
Aadheres pl strat ≡ ∀i. snd (pl i) = strat i (fst (pl i))

isAwinStrat :: [(σ,β) waa, β word, σ Astrat] ⇒ bool
isAwinStrat auto w strat ≡
       isAstrat strat auto w
     ∧ ∀pl ∈ plays auto w. Aadheres pl strat ⇒ Awins auto pl
```

We have now assembled enough definitions to test them by proving some theorems. More constructions will be given as we go along.

4 Acceptance and winning strategies

The first subgoal is to establish the following lemma, reproduced from [9]:

Proposition 2. *The weak alternating automaton $\mathcal{A}$ accepts w iff player A has a local winning strategy in the game associated with $\mathcal{A}$ and w.*

The proof of the "only if" part requires the definition of a winning strategy for player A, given an accepting run (dag) of $\mathcal{A}$ over w. The idea is to let A force the outcome of the play to be a path in the given dag by choosing, for any given position, the successors of that state in the dag. Formally, we enter the goal

```
⟦ dg ∈ run_dags auto w; is_accepting auto dg ⟧
⟹ ∃ strat. isAwinStrat auto w strat
```

For the proof, we simply provide the "witness" term `succs dg` for the existential quantifier. Isabelle's automatic tactics are able to solve the resulting subgoal by expanding the necessary definitions.

For the other direction we must construct an accepting dag, given a winning strategy for player A. Again, we simply let the successors in the dag be defined by the strategy and prove that the dag

```
  (| root = initial auto, succs = strat |)
```

is accepting if `strat` is a winning strategy for player A. The machine proof follows the same pattern outlined above. Note that the concise form of these statements and the natural proof pattern is possible because we are working in a higher-order setting.

5 Dualizing automata and strategies

Encouraged by the success of the proof of the first subproblem, we continue to follow Thomas' exposition. His second lemma connects strategies for an automaton $\mathcal{A}$ with those for its "dual" automaton $\tilde{A}$ obtained by dualizing the transition relation and incrementing the ranks.

```
dual :: (σ,β) waa ⇒ (σ,β) waa
dual auto ≡ (| states = states auto,
                initial = initial auto,
                trans   = λ s b. dual_form (trans auto s b),
                rank    = λ s. (rank auto s)+1 |)
```

It is easy to see (and proved automatically by Isabelle) that $\tilde{\mathcal{A}}$ is well-formed if $\mathcal{A}$ is.

Our main goal is to show that winning strategies for player A in the game associated with automaton $\mathcal{A}$ give rise to winning strategies for P in the game for $\tilde{A}$ and vice versa, as asserted by the following proposition taken from Thomas' paper:

Proposition 4. *Player A has a local winning strategy in the game associated with $\mathcal{A}$ and word w iff player P has a local winning strategy in the game for $\tilde{\mathcal{A}}$ and w.*

Looking at Thomas' proof, we find the following description (adapted to our notation) of how to construct a winning strategy for P in the game for $\tilde{\mathcal{A}}$ from a given strategy for A in the game for $\mathcal{A}$:

> Note that in fixing the strategy it suffices to consider only game positions (i, M) which are reachable [...] The set M of the game position (i, M) is produced by player A from a game position (i, s) such that $M \in \widetilde{\delta(s, w_i)}$. Player P chooses such a state s which could produce M via w_i. Now in the game associated with $\mathcal{A}$ at position (i, s), the given local winning strategy of A picks some $R \in \delta(s, w_i)$, and the definition of $\tilde{\delta}$ ensures that there is a state in $R \cap M$. For his move from the game position (i, M), player P chooses such a state [...]

The idea is that player P forces the outcome of the game to be a possible outcome of the game of $\mathcal{A}$; since the minimum rank of that outcome must have been even with $\mathcal{A}$'s ranking function, it will now be odd. However, the construction described by Thomas requires P to take into account the position (i, s) from which player A determined his move in the current draw of the play, which is not allowed by our definition of a local strategy from section 3.3. The first sentence in the quote above might suggest to try an inductive definition, but this also fails. We therefore revise the definition of type `Pstrat` and the corresponding well-formedness predicate as follows:

```
types
  σ Pstrat = [nat, σ Apos, σ Ppos] ⇒ σ Apos
```

```
isPstrat :: [σ Pstrat, (σ,β) waa, β word] ⇒ bool
isPstrat strat auto w ≡
∀i s M. M ∈ Amoves auto w i s ⇒ strat i s M ∈ Pmoves auto w i M
```

With this revised definition, the construction becomes a direct transcription of Thomas' description: player P chooses some state that is both in the set of states chosen by A and in the set the original strategy would have chosen from $\mathcal{A}$'s transition function; such a state must exist by the relationship between the models of a positive Boolean formula and its dual. In Isabelle this construction is defined by the expression[4]

```
dualizeAstrat :: [(σ,β) waa, β word, σ Astrat] ⇒ σ Pstrat
dualizeAstrat auto w strat ≡ λi s M. εt. t ∈ (M ∩ strat i s)
```

Isabelle proves automatically that the dualized strategy is well-formed if the original strategy is. It remains to prove that it is a winning strategy for P in the game for $\tilde{\mathcal{A}}$ if the original strategy is a winning strategy for A in the game for $\mathcal{A}$. We state the goal

```
isAwinStrat auto w strat
⟹ isPwinStrat (dual auto) w (dualizeAStrategy auto w strat)
```

The main problem is to prove that for every outcome of the dualized strategy there is a play, obtained by applying the original strategy, on the original automaton with the same outcome; that play must therefore be won by A. This proof requires a little guidance and takes 10 interactions.

Thomas asserts that "the other direction is shown analogously, by exchanging the roles of $\mathcal{A}$ and $\tilde{\mathcal{A}}$". Isabelle of course requires an explicit construction, and after the revision of type `Pstrat`, the symmetry is even less obvious. Fortunately, the construction turns out to be quite simple, and there is no need for further modifications to our definitions.

```
dualizePStrat :: [(σ,β) waa, β word, σ Pstrat] => σ Astrat
dualizePStrat auto w strat ≡
    λi s. (strat i s) `` (smodels (trans auto s (w i)))
```

The proof that this construction yields a winning strategy for player A on $\tilde{\mathcal{A}}$ given a winning strategy for P on $\mathcal{A}$ is very similar to the proof described above.

6 Determinacy

The third subproblem in Thomas' proof consists in showing that all games are determined, assuming optimal play.

Proposition 5. *Let $\mathcal{A}$ be a weak alternating automaton. From any position in the game associated with $\mathcal{A}$ and word w, either player A or P has a local winning strategy.*

[4] ε denotes Hilbert's choice operator.

6.1 Attractor sets and associated strategies

The central notion used in the proof is that of an *attractor set*. This is a set of positions (for player A) from which either A or P can force, in finitely many draws, a visit to a set of target positions. Attractor sets for player A are defined inductively as follows:

$$\begin{aligned} attr^0_A(T) &= T \\ attr^{d+1}_A(T) &= attr^d_A(T) \\ &\qquad \cup \{(i,s) \mid \text{for some A-move from } (i,s) \\ &\qquad\qquad \text{all P-moves lead to } attr^d_A(T)\} \end{aligned}$$

The set $attr_A(T)$ is defined as the union of all $attr^d_A(T)$. The definition of $attr_P(T)$ is dual in that $attr^{d+1}_P(T)$ contains those positions for which every A-move leads to a position such that P has some move towards $attr^d_P(T)$. We omit reproducing the formalization of these definitions in Isabelle, which is straightforward.

Thomas writes: "From the positions in $attr_A(T)$ player A can force a decrease of distance to T in each step (which defines a local strategy)". Let us make this explicit:[5]

```
attrA_strat :: [(σ,β)waa, β word, (nat * σ Apos)set] ⇒ σ Astrat
attrA_strat auto w T ≡
λi s. εM. M ∈ Amoves auto w i s
        ∧ ∀d. (i,s) ∈ pick_ea (Amoves auto w) (Pmoves auto w)
                                                (attrAh d auto w T)
              ⇒ ∀t∈M. ∃e. e ≤ d
                          ∧ (i+1,t) ∈ attrAh e auto w T
```

It is easy to prove that this is a well-defined strategy, although the quantifiers are now nested deeply enough for Isabelle to require some guidance. It remains to show that player A can force, via this strategy, a visit in the set T of target positions:

```
⟦ pl ∈ drawseqs auto w s; (i, fst (pl i)) ∈ attrA auto w T;
  ∀j. i ≤ j ⇒ snd (pl j) = attrA_strat auto w T j (fst (pl j)) ⟧
⟹ ∃j. (i+j, fst (pl (i+j))) ∈ T
```

This assertion is proved by a rather straightforward induction on the distance d in the definition of $attr^d_A(T)$. More interesting is the formalization of the next sentence in Thomas' proof: "Also note that for any game position outside $attr_A(T)$, player P will be able to avoid entering this set [...]; otherwise the position would be already in $attr_A(T)$ itself. So from outside $attr_A(T)$, P can avoid, by a local strategy, to enter this set and can hence avoid the visit of T." We boldly formulate a strategy (for P) that avoids entering $attr_A(T)$:

[5] The auxiliary function `pick_ea` formalizes the body of the definition of $attr^{d+1}_A(T)$.

```
CattrA_strat :: [(σ,β) waa, β word, (nat * σ Apos) set]
                ⇒ σ Pstrat
CattrA_strat auto w T ≡
λi s M. εt. t ∈ M
        ∧ ((i,s) ∉ attrA auto w T ⇒ (i+1,t) ∉ attrA auto w T)
```

Why is this a well-defined strategy? Assume that $(i+1,t) \in attr_A(T)$ holds for all possible moves t from M; that is, for every such t there exists some d such that $(i+1,t) \in attr_A^d(T)$. Assuming that $M \neq \emptyset$ was a legal A-move from state s, we need to prove that this implies $(i,s) \in attr_A(T)$. It is here that we require M to be a finite set, for then there is some d such that $(i+1,t) \in attr_A^d(T)$ for all $t \in M$, and therefore we obtain $(i,s) \in attr_A^{d+1}(T) \subseteq attr_A(T)$. Without this finiteness assumption, we would only be able to prove $(i,s) \in pick\_ea(attr_A(T))$, which need not be a subset of $attr_A(T)$.

It is only at this point in the proof (and in the similar proof for player P) where finiteness plays a role, and we therefore obtain a slight generalization of Thomas' result in that the set of states need not be finite, although the transition relations must be defined by finitary formulas.

With this observation, the required well-formedness and correctness conditions are not too hard to establish. In fact, their proofs are made more perspicuous by observing the following algebraic facts about attractor sets.

```
T ⊆ U ⟹ attrA auto w T ⊆ attrA auto w U
is_waa auto ⟹ attrA auto w (attrA auto w T) = attrA auto w T
```

The definitions and proofs for strategies forcing or avoiding visits to $attr_P(T)$ can mostly be obtained by cut-and-paste: the same tactics can also handle the dual quantifier combinations.

6.2 Proving determinacy

Completing the proof of proposition 5, Thomas lets Q_i denote the set of states with rank i and Pos_0 the set of all positions in the game for automaton $\mathcal{A}$ and word w, and continues:

> Clearly, from the positions in $A_0 := attr_A(Q_0)$, Automaton can force, by a local strategy, to reach states of rank 0 and thus win. Consider the subgame whose set of positions is $Pos_1 := Pos_0 \setminus A_0$ (all of which have rank ≥ 1). From the positions in $A_1 := attr_P(Pos_1 \cap Q_1)$, Pathfinder can force, again by a local strategy to reach (and stay in) states of rank 1 and hence win. [...] In this way we continue: In the game with position set $Pos_2 := Pos_1 \setminus A_1$ (containing only states of rank ≥ 2) we form the attractor set $A_2 := attr_A(Pos_2 \cap Q_2)$, etc. Then the positions from which Automaton wins (by the local attractor strategies) are those in the sets A_i with even i. Similarly, Pathfinder wins from the positions in the set A_i with odd i (again by his local attractor strategies).

Instead of formalizing the induction principle based on "subgames" that this argument is based on, we give a direct recursive definition of the sets A_i:[6]

[6] Our definition ensures `determ_pos i` $\subseteq$ `determ_pos (i+2)`, unlike Thomas' definition.

```
rankSet :: [σ ⇒ nat, nat] ⇒ (nat * σ) set
rankSet rk r ≡ { (i,s) | rk s = r }

determ_pos :: [nat, (σ,β) waa, β word] ⇒ (nat * σ Apos) set
recdef determ_pos "less_than"
  determ_pos 0 = λauto w. attrA auto w (rankSet (rank auto) 0)
  determ_pos 1 = λauto w. attrP auto w ((rankSet (rank auto) 1)
                                   \ (determ_pos 0 auto w))
  determ_pos (Suc (Suc k)) = λauto w.
    let pos = determ_pos k auto w
              ∪ ((rankSet (rank auto) (Suc (Suc k)))
                  \ (determ_pos (Suc k) auto w))
    in
    if (k mod 2 = 0) then attrA auto w pos else attrP auto w pos
```

We must also define the strategy player A should apply for positions in the set A_{2k}.

```
A_strat :: [nat, (σ,β) waa, β word] ⇒ σ Astrat
primrec
  A_strat 0 auto w =
  λi s. if (i,s) ∈ rankSet (rank auto) 0
        then εS. S ∈ Amoves auto w i s
        else attrA_strat auto w (rankSet (rank auto) 0) i s

  A_strat (Suc k) auto w =
  λi s. if (i,s) ∈ determ_pos (k+k) auto w
        then A_strat k auto w i s
        else if (i,s) ∈ rankSet (rank auto) (Suc (Suc (k+k)))
                         \ (determ_pos (Suc (k+k)) auto w)
        then CattrP_strat auto w (determ_pos(Suc(k+k)) auto w) i s
        else attrA_strat auto w
                  (determ_pos (k+k) auto w
                   ∪ (rankSet (rank auto) (Suc (Suc (k+k)))
                       \ (determ_pos (Suc (k+k)) auto w))) i s
```

These definitions may look intimidating, but the idea is quite simple: given a position (i, s) in $A_0 = attr_A(Q_0)$, either the rank of s is already 0, in which case any continuation ensures that A wins. Otherwise, player A applies the attractor strategy that is known to force a visit in Q_0. Given a position $(i, s) \in A_{2k+2}$ there are three cases: either in fact $(i, s) \in A_{2k}$ holds, and A applies the strategy defined for positions in A_{2k}, which is already known by induction hypothesis. Second, if $(i, s) \in Q_{2k+2} \setminus A_{2k+1}$ (the "kernel" of A_{2k+2}) then A avoids positions in A_{2k+1}; this ensures that the ranks of subsequent positions will be even and $\leq 2k + 2$. Otherwise, player A applies the attractor strategy for A_{2k+2} to force an eventual visit in the kernel.

Continuing the formal development, we first prove that every position is contained in some A_i:

```
∃k. k ≤ rank auto s ∧ (i,s) ∈ determ_pos k auto w
```

The proof is by case distinction on whether the rank r of s is 0, 1, or $i+2$ for some $i \in \mathbb{N}$, expansion of the appropriate clause in the definition of A_r, and further case distinctions guided by the definitions, taking 13 interactions. More difficult is the proof of the following lemma, which asserts that strategy `A_strat`, applied to some position in A_{2k} forces the play to remain in some position in A_{2l}, for some $l \leq k$.

```
⟦ is_waa auto; pl ∈ drawseqs auto w s;
  (i,fst (pl i)) ∈ determ_pos (k+k) auto w;
  snd (pl i) = A_strat k auto w i (fst (pl i)) ⟧
⟹ ∃l. l ≤ k ∧ (i+1,fst(pl(i+1))) ∈ determ_pos (l+l) auto w
```

The proof is essentially by induction on k, but requires 44 interactions, including many low-level instantiations. Of similar complexity is the proof of the overall correctness of the strategy, which takes 34 interactions.

```
⟦ is_waa auto; pl ∈ drawseqs auto w s;
  (i, fst (pl i)) ∈ determ_pos (k+k) auto w;
  ∀j. i ≤ j ⇒ snd (pl j) = A_strat k auto w j (fst (pl j)) ⟧
⟹ Awins auto pl
```

It remains to define a similar winning strategy for player P when starting from a position in A_{2k+1}, and reprove the analogous theorems. The basic ideas are the same, but due to the low-level nature of some of the proof scripts, they cannot be copied blindly. Since obviously there cannot be a winning strategy for both players for some position, it follows that the sets A_k form two disjoint hierarchies for k even or odd.

6.3 Complementation

We can now prove the complementation theorem:

```
is_waa auto ⟹ language (dual auto) = - language auto
```

The proof is simple, given the previous lemmas: automaton $\mathcal{A}$ does not accept input word w iff (by proposition 2) player A does not have a local winning strategy in the corresponding game iff (by proposition 4) player P does not have a local winning strategy in the game associated with $\tilde{\mathcal{A}}$ and w iff (by proposition 5) player A has a local winning strategy in that game iff (again by proposition 2) $\tilde{\mathcal{A}}$ accepts w.

7 From Temporal Logic to Weak Alternating Automata

As an application of our definitions and results, we now formalize a translation of formulas of propositional temporal logic of linear time into weak alternating automata [4, 11]. In contrast to the standard translation of PTL to Büchi automata, this construction is compositional and of linear complexity; the hardest step in the correctness proof is that for negation, which is based on the theorem we have just proven.

7.1 Propositional Temporal Logic

We begin with the definition of a deep embedding of PTL (over a type α of atomic propositions) in Isabelle:[7]

```
datatype α ptl =
    TRUE | Var α | NOT α ptl | OR (α ptl) (α ptl)
  | NEXT α ptl  | UNTIL (α ptl) (α ptl)
type α behavior = (α set) word

suffix :: [α behavior, nat] ⇒ α behavior
suffix ρ n ≡ λi. ρ(i+n)

holdsAt :: [α behavior, α ptl] ⇒ bool   ("_ ⊨ _")
ρ ⊨ TRUE
ρ ⊨ Var x     =  (x ∈ ρ 0)
ρ ⊨ NOT f     =  ¬(ρ ⊨ f)
ρ ⊨ f OR g    =  ((ρ ⊨ f) ∨ (ρ ⊨ g))
ρ ⊨ NEXT f    =  suffix ρ 1 ⊨ f
ρ ⊨ f UNTIL g =  ∃n. suffix ρ n ⊨ g ∧ ∀ m<n. suffix ρ m ⊨ f

models :: α ptl ⇒ (α behavior) set
models f ≡ { ρ | ρ ⊨ f }
```

We will mainly restrict attention to *normal* formulas that do not contain double negations; this predicate is easily defined as a recursive function, as is a normalization function that removes any double negations. We can then prove laws about PTL formulas such as

```
(ρ ⊨ f UNTIL g) = ((ρ ⊨ g) ∨ (ρ ⊨ f ∧ ρ ⊨ NEXT(f UNTIL g)))
```

We inductively define the set of subformulas of a PTL formula. The *Fischer-Ladner closure* $\mathcal{C}(\varphi)$ of a formula φ is the set that contains all subformulas of φ and their complements. Observe that all formulas in $\mathcal{C}(\varphi)$ are normal if φ is normal.

[7] We use infix syntax for the `OR` and `UNTIL` operators.

7.2 Definition of Automaton $\mathcal{A}_\varphi$

We associate a weak alternating automaton $\mathcal{A}_\varphi$ with every PTL formula φ. The set of states is given by the Fischer-Ladner closure of φ, with φ being the initial state. The transition relation and the ranks are defined by induction on the formula.[8]

```
ptl_trans :: [α ptl, α set] ⇒ (α ptl) pboolean
ptl_trans TRUE S = Atomic TRUE
ptl_trans (Var x) S =
    if x∈s then Atomic TRUE else Atomic(NOT TRUE)
ptl_trans (NOT f) S = subst complement (dual_form(ptl_trans f S))
ptl_trans (f OR g) S = Or (ptl_trans f S) (ptl_trans g S)
ptl_trans (NEXT f) S = Atomic f
ptl_trans (f UNTIL g) s =
    Or (ptl_trans g s)
       (And (Atomic (f until g)) (ptl_trans f S))

ptl_rank :: α ptl ⇒ nat
ptl_rank TRUE = 0
ptl_rank (Var x) = 1
ptl_rank (NOT f) = (ptl_rank f)+1
ptl_rank (f OR g) = max (ptl_rank f) (ptl_rank g)
ptl_rank (NEXT f) = ptl_rank f
ptl_rank (f UNTIL g) = let r = max (ptl_rank f) (ptl_rank g)
                       in  if r mod 2 = 0 then r+1 else r

ptl_waa :: α ptl ⇒ (α ptl, α set) waa
ptl_waa f ≡
     (| states  = fischer_ladner f, initial = f,
        trans   = ptl_trans,        rank    = ptl_rank |)
```

The definition of the transition relation for an automaton state corresponding to an **until** formula follows the recursive expansion law shown above. For a formula $\neg\varphi$, we take the dual of the Boolean combination associated with φ, but complement all atoms. Based on a few lemmas about the Fischer-Ladner closure, it is easy to prove that automaton $\mathcal{A}_\varphi$ is indeed well-formed if φ is a normal PTL formula.

```
normal f ⟹ is_waa (ptl_waa f)
```

Moreover, the automaton associated with the complement of φ is isomorphic to the dual automaton for φ modulo a renaming of all its states by complementation and possibly an adjustment of ranks that does not affect their parity.

[8] `subst h p` denotes the positive Boolean formula obtained from `p` by applying `h` to all atoms in `p`.

```
ren_auto  :: [(σ,β) waa, σ  ⇒  τ, (τ,β) waa] ⇒ bool
ren_auto src h tgt ≡
   states tgt = h``(states src)
 ∧ initial tgt = h(initial src)
 ∧ ∀s∈states src. ∀b. trans tgt (h s) b = subst h (trans src s b)
 ∧ ∀s∈states src. rank tgt (h s) mod 2 = rank src s mod 2

normal f ⟹
ren_auto (dual (ptl_waa f)) complement (ptl_waa (complement f))
```

7.3 Correctness of the Translation

We now prove that the automaton $\mathcal{A}_\varphi$ accepts precisely the models of φ, for any normal PTL formula φ. This requires proving that (i, ψ) is a winning position for player A in the game associated with $\mathcal{A}_\varphi$ and behavior ρ if and only if ψ holds of the i'th suffix of ρ, for any formula ψ in the Fischer-Ladner closure of φ. The only non-trivial cases are those for negation and **until**. For negation, the assertion follows by the complementation theorem for weak alternating automata, the observation that $\mathcal{A}_{\neg\psi}$ is obtained by renaming $\mathcal{A}_\psi$ via complementation, and the fact that complementation is injective for normal formulas. The most tedious part of the Isabelle proof is to show that an automaton obtained by renaming via a function that is injective on the states of the original automaton induces the same winning positions for either player. For a formula $\psi \equiv \psi_1$ **until** ψ_2, it suffices to observe that any draw sequence starting at ψ remains at ψ until it simulates a draw sequence for ψ_1 or ψ_2. Using the recursion law for the **until** operator, this justifies the definition of a winning strategy for player A.

Assembling the bits and pieces, we can now prove the correctness of the translation from PTL formulas to weak alternating automata:

```
language (ptl_waa (normalize f)) = models f
```

Isabelle's facility to state goals containing unknowns, which are instantiated during the proof, allows us to use this theorem to actually construct automata $\mathcal{A}_\varphi$ inside the theorem prover, based on Isabelle's built-in rewriting engine. In this way, we obtain a certified translation procedure from PTL formulas to weak alternating automata. The evaluation can actually be performed inside Isabelle because the construction is of linear complexity.

8 Conclusion

The work on this paper has been started by pure intellectual curiosity: Complementation of ω-automata has been one of the most celebrated problems of the field, and Thomas' streamlined presentation of the proof based on weak alternating automata and their associated games seemed to be the first one that could actually be carried out using an interactive theorem prover. The usefulness of such an endeavor may, however, be disputable. This case study has once again confirmed that at least part of the social process that leads to the acceptance of a proof can actually be replaced by machine certification, and that even well-polished hand proofs are still likely to contain errors such as the "type error" our formalization has revealed.

On the other hand, it cannot be disputed that it still takes some effort to convert even a very well-presented proof into machine-checkable form: our formalization of Thomas' proof took about three weeks, being reasonably familiar with both the subject matter and the tool. Still, this is perhaps not an inordinate amount of time for a reasonably complex proof.

It is obvious that formalization of mathematical proofs requires attention to details that are happily ignored in paper-and-pencil proofs. Still, if formalization in itself is to be of any value, the format of the documents produced by machine proofs is also important when discussing formalized mathematics. In this respect, our experience has been mixed. On the one hand, we found that the formalization of the underlying concepts was fairly natural and simple, due to the high expressiveness of the higher-order framework. Concerning the automation of proofs, the first part of the work reported here was carried out using Isabelle's standard tactic-based interface. We have found that the automatic tactics provided by Isabelle are incompatible with human reasoning capabilities: whereas complicated-looking subgoals may succumb to a single tactic invocation, other seemingly trivial goals (in particular those that involve arithmetic) need tedious low-level interaction. Moreover, small changes to the underlying definitions may render a tactic script obsolete. For the translation of PTL to automata, we have used Wenzel's recent Isar interface [12] that is based on an explicit proof language. This has been a very positive experience: although documents become much more verbose, the user has much more control of the interaction, and changes tend to be much easier to apply.

The main benefit of formalization, however, is exemplified by the certified translation of PTL formulas to automata presented in section 7 where effective computational procedures have been derived from the purely mathematical constructions arising from the proof. Our translation to automata constitutes the first step towards a formalized PTL decision procedure: having constructed the automaton, it remains to decide whether its language is empty or not in order to decide whether the formula is satisfiable or not. Deciding nonemptiness is rather easy using external tools ("oracles" in Isabelle terminology), for example based on BDD technology. In general, deciding nonemptiness of weak alternating automata is PSPACE-complete, and it is therefore less clear how to carry out this second step inside a theorem prover. Schneider [6] has presented a similar translation of PTL formulas to a symbolic representation of Büchi automata. His translation is also linear, and he also uses an external tool to decide nonemptiness. We believe that alternating automata may be preferable to Büchi automata because the translation is compositional, which may be an advantage for applications such as model checking. For example, one may want to apply minimizations during the construction of $\mathcal{A}_\varphi$, or pre-compute the product of the transition system under investigation with, say, certain fairness conditions.

References

1. J. R. Büchi. On a decision method in restricted second-order arithmetics. In *International Congress on Logic, Method and Philosophy of Science*, pages 1–12. Stanford University Press, 1962.
2. Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not so weak. In *5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Press, 1997.
3. D. E. Muller, A. Saoudi, and P. E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *13th ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 275–283. Springer-Verlag, 1986.
4. D.E. Muller, A. Saoudi, and P.E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *3rd IEEE Symposium on Logic in Computer Science*, pages 422–427. IEEE Press, 1988.
5. Shmuel Safra. On the complexity of ω-automata. In *29th IEEE Symposium on Foundations of Computer Science*, pages 319–327. IEEE Press, 1988.
6. Klaus Schneider and Dirk W. Hoffmann. A HOL conversion for translating linear time temporal logic to ω-automata. In Y. Bertot et al, editor, *TPHOLs'99: 12th International Conference on Theorem Proving in Higher Order Logics*, volume 1690 of *Lecture Notes in Computer Science*, pages 255–272, Nice, France, 1999. Springer-Verlag.
7. Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics*, pages 133–194. Elsevier, Amsterdam, 1990.
8. Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer-Verlag, New York, 1997.
9. Wolfgang Thomas. Complementation of Büchi automata revisited. In J. Karhumäki, editor, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–122. Springer-Verlag, 2000.
10. Moshe Vardi. Verification of concurrent programs: The automata-theoretic framework. In *Proceedings of the Second Symposium on Logic in Computer Science*, pages 167–176. IEEE, June 1987.
11. Moshe Y. Vardi. *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, chapter Alternating Automata and Program Verification, pages 471–485. Springer-Verlag, 1995.
12. Markus Wenzel. Isar—a generic interpretative approach to readable formal proof documents. In Y. Bertot et al, editor, *TPHOLs'99: 12th International Conference on Theorem Proving in Higher Order Logics*, volume 1690 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.