We present a formalization of parity games (a two-player game on directed graphs) and a proof of their positional determinacy in Isabelle/HOL. This proof works for both finite and infinite games. We follow the proof in [2], which is based on [3].
12.2 Positional Determinacy without Deadends ........................................ 34
12.3 Positional Determinacy with Deadends ........................................... 34
12.4 The Main Theorem: Positional Determinacy ........................................ 34

13 Defining the Attractor with inductive_set ........................................... 34
  13.1 attractor-inductive ............................................................................. 34

Bibliography .............................................................................................. 36
1 Introduction

Parity games are games played by two players, called Even and Odd, on labelled directed graphs. Each node is labelled with their player and with a natural number, called its priority.

To call this a parity game, we only need to assume that the number of different priorities is finite. Of course, this condition is only relevant on infinite graphs.

One reason parity games are important is that determining the winner is polynomial-time equivalent to the model-checking problem of the modal $\mu$-calculus, a logic able to express LTL and CTL* properties ([1]).

1.1 Formal Introduction

Formally, a parity game is $G = (V, E, V_0, \omega)$, where $(V, E)$ is a directed graph, $V_0 \subseteq V$ is the set of Even nodes, and $\omega : V \rightarrow \mathbb{N}$ is a function with $|f(V)| < \infty$.

A play is a maximal path in $G$. A finite play is winning for Even iff the last node is not in $V_0$. An infinite play is winning for Even iff the minimum priority occurring infinitely often on the path is even. On an infinite path at least one priority occurs infinitely often because there is only a finite number of different priorities.

A node $v$ is winning for a player $p$ iff all plays starting from $v$ are winning for $p$. It is well-known that parity games are determined, that is, every node is winning for some player.

A more surprising property is that parity games are also positionally determined. This means that for every node $v$ winning for Even, there is a function $\sigma : V_0 \rightarrow V$ such that all Even needs to do in order to win from $v$ is to consult this function whenever it is his turn (similarly if $v$ is winning for Odd). This is also called a positional strategy for the winning player.

We define the winning region of player $p$ as the set of nodes from which player $p$ has positional winning strategies. Positional determinacy then says that the winning regions of Even and of Odd partition the graph.

See [3] for a modern survey on positional determinacy of parity games. Their proof is based on a proof by Zielonka [5].

1.2 Overview

Here we formalize the proof from [2] in Isabelle/HOL. This proof is similar to the proof in [3], but we do not explicitly define so-called “$\sigma$-traps”. Using $\sigma$-traps could be worth exploring, because it has the potential to simplify our formalization.

Our proof has no assumptions except those required by every parity game. In particular the parity game

- may have arbitrary cardinality,
- may have loops,
- may have deadends, that is, nodes with no successors.

The main theorem is in section 12.4.
1.3 Technical Aspects

We use a coinductive list of nodes to represent paths in a graph because this gives us a uniform representation for finite and infinite paths. We can then express properties such as that a path is maximal or conforms to a given strategy directly as coinductive properties. We use the coinductive list developed by Lochbihler in [4].

We also explored representing paths as functions $nat \Rightarrow 'a option$ with the property that the domain is an initial segment of $nat$ (and where $'a$ is the node type). However, it turned out that coinductive lists give simpler proofs.

It is possible to represent a graph as a function $'a \Rightarrow 'a \Rightarrow bool$, see for example in the proof of König’s lemma in [4]. However, we instead go for a record which contains a set of nodes and a set of edges explicitly. By not requiring that the set of nodes is $UNIV :: 'a set$ but rather a subset of $UNIV :: 'a set$, it becomes easier to reason about subgraphs.

Another point is that we make extensive use of locales, in particular to represent maximal paths conforming to a specific strategy. Thus proofs often start with interpret $vmc-path G P v_0 p \sigma$ to say that $P$ is a valid maximal path in the graph $G$ starting in $v_0$ and conforming to the strategy $\sigma$ for player $p$.

2 Auxiliary Lemmas for Coinductive Lists

Some lemmas to allow better reasoning with coinductive lists.

theory MoreCoinductiveList
imports
  Main
  ../Coinductive/Coinductive-List
begin

2.1 lset

lemma lset-lnth: $x \in lset xs \Rightarrow \exists n. \text{lnth } xs n = x$
  (proof)

lemma lset-lnth-member: $[ lset xs \subseteq A; \text{enat } n < llength xs ] \Rightarrow \text{lnth } xs n \in A$
  (proof)

lemma lset-nth-member-inf: $[ \neg lfinite xs; lset xs \subseteq A ] \Rightarrow \text{lnth } xs n \in A$
  (proof)

lemma lset-intersect-lnth: $lset xs \cap A \neq \{\} \Rightarrow \exists n. \text{enat } n < llength xs \land \text{lnth } xs n \in A$
  (proof)

lemma lset-ltake-Suc:
  assumes $\neg lnull xs \text{lnth } xs 0 = x lset (ltake (\text{enat } n) (\text{ltl } xs)) \subseteq A$
  shows $lset (ltake (\text{enat } (\text{Suc } n)) xs) \subseteq \text{insert } x A$
  (proof)

lemma lfinite-lset: $lfinite xs \Rightarrow \neg lnull xs \Rightarrow llast xs \in lset xs$
  (proof)
lemma lset-subset: \(\neg (lset \; xs \subseteq A) \implies \exists \; n. \; \text{enat} \; n < \text{llength} \; xs \land \text{lnth} \; xs \; n \notin A\)

(proof)

2.2 llength

lemma enat-Suc-ltl:
  assumes enat \((\text{Suc} \; n) < \text{llength} \; xs\)
  shows enat \(n < \text{llength} \; (\text{ltl} \; xs)\)

(proof)

lemma enat-ltl-Suc:
  enat \(n < \text{llength} \; (\text{ltl} \; xs)\) =\(\implies\) enat \((\text{Suc} \; n) < \text{llength} \; xs\)

(proof)

lemma infinite-small-llength [intro]: \(\neg \text{lfinite} \; xs \implies \text{enat} \; n < \text{llength} \; xs\)

(proof)

lemma lnull-0-llength: \(\neg \text{lnull} \; xs \implies \text{enat} \; 0 < \text{llength} \; xs\)

(proof)

lemma Suc-llength:
  enat \((\text{Suc} \; n) < \text{llength} \; xs\) =\(\implies\) enat \(n < \text{llength} \; xs\)

(proof)

2.3 ltake

lemma ltake-lnth:
  \(\text{ltake} \; n \; xs = \text{ltake} \; n \; ys \implies \text{enat} \; m < n \implies \text{lnth} \; xs \; m = \text{lnth} \; ys \; m\)

(proof)

lemma lset-ltake-prefix [simp]: \(n \leq m \implies lset \; (\text{ltake} \; n \; xs) \subseteq lset \; (\text{ltake} \; m \; xs)\)

(proof)

lemma lset-ltake:
  \(\bigwedge m. \; m < n \implies \text{lnth} \; xs \; m \in A \implies lset \; (\text{ltake} \; (\text{enat} \; n) \; xs) \subseteq A\)

(proof)

lemma llength-ltake':
  enat \(n < \text{llength} \; xs\) =\(\implies\) llength \((\text{ltake} \; (\text{enat} \; n) \; xs)\) = enat \(n\)

(proof)

lemma llast-ltake:
  assumes enat \((\text{Suc} \; n) < \text{llength} \; xs\)
  shows llast \((\text{ltake} \; (\text{enat} \; (\text{Suc} \; n)) \; xs)\) = lnth \; xs \; n \; (\text{is} \; \text{llast} \; ?A = -)\)

(proof)

lemma lset-ltake-ltl:
  lset \; (\text{ltake} \; (\text{enat} \; n) \; (\text{ltl} \; xs)) \subseteq lset \; (\text{ltake} \; (\text{enat} \; (\text{Suc} \; n)) \; xs)\)

(proof)

2.4 ldropn

lemma ltl-ldrop:
  \(\bigwedge xs. \; P \; xs \implies P \; (\text{ltl} \; xs); \; P \; xs \implies P \; (\text{ldropn} \; n \; xs)\)

(proof)

2.5 lfinite

lemma lfinite-drop-set:
  lfinite \; xs \implies \exists n. \; v \notin lset \; (\text{ldrop} \; n \; xs)\)
lemma index-infinite-set:
\[
\neg \text{finite } x; \lnth x m = y; \forall i. \lnth x i = y \implies (\exists m > i. \lnth x m = y) \] \implies y \in lset (ldropn n x)
\]

2.6 \textit{lmap}

lemma lmap-lmap-ldropn:
\[
\text{enat } n < \text{llength } xs \implies \lnth (\text{lmap } f \ (\text{ldropn } n \ xs)) \ 0 = \lnth (\text{lmap } f \ xs) \ n
\]

lemma lmap-lmap-ldropn-Suc:
\[
\text{enat } (\text{Suc } n) < \text{llength } xs \implies \lnth (\text{lmap } f \ (\text{ldropn } n \ xs)) \ (\text{Suc } 0) = \lnth (\text{lmap } f \ xs) \ (\text{Suc } n)
\]

2.7 Notation

We introduce the notation \$ to denote \lnth.

totality \lnth (\text{infix } 61)

end

3 Parity Games

theory ParityGame
imports
  Main
  MoreCoinductiveList
begin

3.1 Basic definitions
\'a is the node type. Edges are pairs of nodes.

type-synonym \'a Edge = \'a \times \'a

A path is a possibly infinite list of nodes.

type-synonym \'a Path = \'a llist

3.2 Graphs

We define graphs as a locale over a record. The record contains nodes (AKA vertices) and edges. The locale adds the assumption that the edges are pairs of nodes.

record \'a Graph =
  verts :: \'a set (V1)
  arcs :: \'a Edge set (E1)
abbreviation is-arc :: \('a, 'b Graph-scheme ⇒ \'a ⇒ \'a ⇒ bool (\text{infixl } \rightarrow 60) \) where
  \( v \rightarrow_G w \equiv (v,w) \in E_G \)
locale Digraph = 
  fixes G (structure)
  assumes valid-edge-set: E ⊆ V × V
begin
lemma edges-are-in-V [intro]: v→w ⇒ v ∈ V v→w ⇒ w ∈ V ⟨proof⟩

A node without successors is a deadend.
abbreviation deadend :: 'a ⇒ bool where deadend v ≡ ¬(∃ w ∈ V. v → w)

3.3 Valid Paths
We say that a path is valid if it is empty or if it starts in V and walks along edges.

coinductive valid-path :: 'a Path ⇒ bool where
  valid-path-base: valid-path LNil
  | valid-path-cons: [ v ∈ V; w ∈ V; v→w; valid-path Ps; ¬lnull Ps; lhd Ps = w ]
  ⇒ valid-path (LCons v Ps)

inductive-simps valid-path-cons-simp: valid-path (LCons x xs)

lemma valid-path-ltl': valid-path (LCons v Ps) ⇒ valid-path Ps ⟨proof⟩
lemma valid-path-ltl: valid-path P ⇒ valid-path (ltl P) ⟨proof⟩
lemma valid-path-drop: valid-path P ⇒ valid-path (ldropn n P) ⟨proof⟩

lemma valid-path-in-V: assumes valid-path P shows lset P ⊆ V ⟨proof⟩
lemma valid-path-finite-in-V: [ valid-path P; enat n < llength P ] ⇒ P $ n ∈ V ⟨proof⟩

lemma valid-path-edges': valid-path (LCons v (LCons w Ps)) ⇒ v→w ⟨proof⟩
lemma valid-path-edges:
  assumes valid-path P enat (Suc n) < llength P
  shows P $ n → P $ Suc n ⟨proof⟩

lemma valid-path-coinduct [consumes 1, case-names base step, coinduct pred: valid-path]:
  assumes major: Q P
  and base: λ v P. Q (LCons v LNil) ⇒ v ∈ V
  and step: λ v w P. Q (LCons v (LCons w P)) ⇒ v→w ∧ (Q (LCons w P) ∨ valid-path (LCons w P))
  shows valid-path P ⟨proof⟩

lemma valid-path-no-deadends:
  [ valid-path P; enat (Suc i) < llength P ] ⇒ ¬deadend (P $ i) ⟨proof⟩
lemma valid-path-ends-on-deadend:
\[
[ \text{valid-path } P; \text{enat } i < \text{llength } P; \text{deadend } (P \mid i) ] \implies \text{enat } (\text{Suc } i) = \text{llength } P
\]
(proof)

lemma valid-path-prefix: \[
[ \text{valid-path } P; \text{lprefix } P' P ] \implies \text{valid-path } P'
\]
(proof)

lemma valid-path-lappend:
assumes valid-path P valid-path P' \[
\neg \text{lnull } P; \neg \text{lnull } P'
\]
shows valid-path (lappend P P')
(proof)

A valid path is still valid in a supergame.

lemma valid-path-supergame:
assumes valid-path P and \( G' : \text{Digraph } G' V \subseteq V_G E \subseteq E_G \)
shows Digraph.valid-path G' P
(proof)

3.4 Maximal Paths

We say that a path is maximal if it is empty or if it ends in a deadend.

coinductive maximal-path where
\[
\begin{align*}
\text{maximal-path-base: } & \text{maximal-path } \text{LNil} \\
\text{maximal-path-base': } & \text{deadend } v \implies \text{maximal-path } (\text{LCons } v \text{LNil}) \\
\text{maximal-path-cons: } & \neg \text{lnull } Ps \implies \text{maximal-path } Ps \implies \text{maximal-path } (\text{LCons } v \text{Ps})
\end{align*}
\]

lemma maximal-no-deadend: maximal-path (LCons v Ps) \implies \neg \text{deadend } v \implies \neg \text{lnull } Ps
(proof)

lemma maximal-ltl: maximal-path P \implies maximal-path (ltl P)
(proof)

lemma maximal-drop: maximal-path P \implies maximal-path (ldropn n P)
(proof)

lemma maximal-path-lappend:
assumes \neg \text{lnull } P' maximal-path P'
shows maximal-path (lappend P P')
(proof)

lemma maximal-ends-on-deadend:
assumes maximal-path P lfinite P \neg \text{lnull } P
shows deadend (llast P)
(proof)

lemma maximal-ends-on-deadend': [ lfinite P; deadend (llast P) ] \implies maximal-path P
(proof)

lemma infinite-path-is-maximal: \[
[ \text{valid-path } P; \neg \text{lfinite } P ] \implies \text{maximal-path } P
\]
(proof)

end — locale Digraph
3.5 Parity Games

Parity games are games played by two players, called EVEN and ODD.

datatype Player = Even | Odd

abbreviation other-player p ≡ (if p = Even then Odd else Even)
notation other-player (p) ≡ (if p = 1 then 1 else 0)
lemma other-other-player [simp]: p === p (proof)

A parity game is tuple \((V, E, V_0, \omega)\), where \((V, E)\) is a graph, \(V_0 \subseteq V\) and \(\omega\) is a function from \(V \to \mathbb{N}\) with finite image.

record ParityGame = Graph +
  player0 :: set (V0)
  priority :: nat (\omega)

locale ParityGame = Digraph G for G :: (\alpha, \beta) ParityGame-scheme (structure) +
  assumes valid-player0-set: V0 \subseteq V
  and priorities-finite: finite (\omega \ V)
begin

VV p is the set of nodes belonging to player \(p\).

abbreviation VV :: Player \to set where VV p ≡ (if p = Even then V \ V0 else V \ V0)
lemma VVp-to-V [intro]: v ∈ VV p \Rightarrow v ∈ V (proof)
lemma VV-impl1: v ∈ VV p \Rightarrow v \notin VV p** (proof)
lemma VV-impl2: v ∈ VV p** \Rightarrow v \notin VV p (proof)
lemma VV-equivalence [iff]: v ∈ V \Rightarrow v \notin VV p \iff v ∈ VV p** (proof)
lemma VV-cases [consumes 1]: [ v ∈ V ; v ∈ VV p \Rightarrow P ; v ∈ VV p** \Rightarrow P ] \Rightarrow P (proof)

3.6 Sets of Deadends

definition deadends p ≡ \{ v ∈ VV p \mid deadend v \}
lemma deadends-in-V: deadends p ⊆ V (proof)

3.7 Subgames

We define a subgame by restricting the set of nodes to a given subset.

definition subgame where
  subgame V' :: G
  verts :: V \cap V',
  arcs :: E \cap (V' \times V'),
  player0 :: V0 \cap V'
lemma subgame-V [simp]: V subgame V' \subseteq V
and subgame-E [simp]: E subgame V' \subseteq E
and subgame-\omega: \omega subgame V' = \omega
(proof)

lemma
  assumes V' \subseteq V
  shows subgame-V' [simp]: V subgame V' = V'
and subgame-E \[\cdot\] simp: \( E_{\text{subgame } V'} = E \cap (V_{\text{subgame } V'} \times V_{\text{subgame } V'}) \)

proof

lemma subgame-VV simp: ParityGame.VV (subgame V') p = V' \cap VV p

proof

corollary subgame-VV-subset simp: ParityGame.VV (subgame V') p \subseteq VV p

proof

lemma subgame-finite simp: finite \((\omega_{\text{subgame } V'} \cdot V_{\text{subgame } V'})\)

proof

lemma subgame-\omega-subset simp: \(\omega_{\text{subgame } V'} \cdot V_{\text{subgame } V'} \subseteq \omega \cdot V\)

proof

lemma subgame-Digraph: Digraph (subgame V')

proof

lemma subgame-ParityGame: shows ParityGame (subgame V')

proof

lemma subgame-valid-path:
  assumes P: valid-path P lset P \subseteq V'
  shows Digraph.valid-path (subgame V') P

proof

lemma subgame-maximal-path:
  assumes V': V' \subseteq V and P: maximal-path P lset P \subseteq V'
  shows Digraph.maximal-path (subgame V') P

proof

3.8 Priorities Occurring Infinitely Often

The set of priorities that occur infinitely often on a given path. We need this to define the winning condition of parity games.

definition path-inf-priorities :: 'a Path \Rightarrow nat set

path-inf-priorities P \equiv \{k. \forall n. k \in \text{lset} (ldropn n (lmap \omega P))}\}

Because \(\omega\) is image-finite, by the pigeon-hole principle every infinite path has at least one priority that occurs infinitely often.

lemma path-inf-priorities-is-nonempty:
  assumes P: valid-path P \neg \text{lfinite} P
  shows \(\exists k. k \in \text{path-inf-priorities} P\)

proof

lemma path-inf-priorities-at-least-min-prio:
  assumes P: valid-path P and a: a \in path-inf-priorities P
  shows Min (\omega \cdot V) \leq a

proof

lemma path-inf-priorities-LCons:
  path-inf-priorities P = path-inf-priorities (LCons v P) (isa ?A = ?B)

proof
corollary path-inf-priorities-ltl: path-inf-priorities $P = \text{path-inf-priorities \ (ltl \ P)}$

\[ (\text{proof}) \]

### 3.9 Winning Condition

Let $G = (V, E, V_0, \omega)$ be a parity game. An infinite path $v_0, v_1, \ldots$ in $G$ is winning for player EVEN (ODD) if the minimum priority occurring infinitely often is even (odd). A finite path is winning for player $p$ iff the last node on the path belongs to the other player.

Empty paths are irrelevant, but it is useful to assign a fixed winner to them in order to get simpler lemmas.

abbreviation winning-priority $p \equiv \begin{cases} \text{even} & \text{if } p = \text{Even} \\ \text{odd} & \text{otherwise} \end{cases}$

definition winning-path :: Player $\Rightarrow$ 'a Path $\Rightarrow$ bool where

\[ \text{winning-path } p \ P \equiv \]

\[ \neg \text{lnull } P \land \left( \exists \ a \in \text{path-inf-priorities } P. \ a \leq b \right) \land \text{winning-priority } p \ a \]

\[ \lor \right( \neg \text{lnull } P \land \left( \text{finite } P \land \text{llast } P \in VV \ p^{**} \right) \]

\[ \lor \left( \text{lnull } P \land p = \text{Even} \right) \]

Every path has a unique winner.

lemma paths-are-winning-for-one-player:

assumes valid-path $P$

shows winning-path $p \ P \iff \neg\text{winning-path } p^{**} \ P$

(proof)

lemma winning-path-ltl:

assumes $P$: winning-path $p \ P \neg \text{lnull } P \neg \text{lnull \ (ltl } P)$

shows winning-path $p \ (\text{ltl } P)$

(proof)

corollary winning-path-drop:

assumes winning-path $p \ P \text{ enat } n < \text{llength } P$

shows winning-path $p \ (ldropn \ n \ P)$

(proof)

corollary winning-path-drop-add:

assumes valid-path $P$ \text{ winning-path } p \ (ldropn \ n \ P) \text{ enat } n < \text{llength } P$

shows winning-path $p \ P$

(proof)

lemma winning-path-LCons:

assumes $P$: winning-path $p \ P \neg \text{lnull } P$

shows winning-path $p \ (\text{LCons } v \ P)$

(proof)

lemma winning-path-supergame:

assumes winning-path $p \ P$

and $G'$: ParityGame $G'$ $VV \ p^{**} \subseteq \text{ParityGame.}VV \ G' \ p^{**} \ \omega = \omega_{G'}$

shows ParityGame.winning-path $G'$ $p \ P$

(proof)
3.10 Valid Maximal Paths

Define a locale for valid maximal paths, because we need them often.

```plaintext
locale vm-path = ParityGame +
  fixes P v0
  assumes P-not-null [simp]: ¬lnull P
  and P-valid [simp]: valid-path P
  and P-maximal [simp]: maximal-path P
  and P-v0 [simp]: lhd P = v0
begin
lemma P-LCons: P = LCons v0 (ltl P) (proof)

lemma P-len [simp]: enat 0 < llength P (proof)
lemma P-0 [simp]: P $ 0 = v0 (proof)
lemma P-lnth-Suc: P $ Suc n = ltl P $ n (proof)
lemma P-no-deadends: enat (Suc n) < llength P ⇒ ¬deadend (P $ n) (proof)
lemma P-no-deadend-v0: ¬lnull (ltl P) ⇒ ¬deadend v0 (proof)
lemma P-no-deadend-v0-llength: enat (Suc n) < llength P ⇒ ¬deadend v0 (proof)
lemma P-ends-on-deadend: [ enat n < llength P; deadend (P $ n) ] ⇒ enat (Suc n) = llength P (proof)
lemma P-lnull-ltl-deadend-v0: lnull (ltl P) ⇒ deadend v0 (proof)
lemma P-lnull-ltl-LCons: lnull (ltl P) ⇒ P = LCons v0 LNil (proof)
lemma P-deadend-v0-LCons: deadend v0 ⇒ P = LCons v0 LNil (proof)
lemma Ptl-valid [simp]: valid-path (ltl P) (proof)
lemma Ptl-maximal [simp]: maximal-path (ltl P) (proof)
lemma Pdrop-valid [simp]: valid-path (ldropn n P) (proof)
lemma Pdrop-maximal [simp]: maximal-path (ldropn n P) (proof)
lemma prefix-valid [simp]: valid-path (ltake n P) (proof)
lemma extension-valid [simp]: v→v0 ⇒ valid-path (LCons v P) (proof)
lemma extension-maximal [simp]: maximal-path (LCons v P) (proof)
lemma lappend-maximal [simp]: maximal-path (lappend P' P) (proof)
lemma v0-V [simp]: v0 ∈ V (proof)
```
lemma \( v_0 \)-lset-P [simp]: \( v_0 \in \text{lset } P \) (proof)
lemma \( v_0 \)-VV: \( v_0 \in \text{VV } p \lor v_0 \in \text{VV } p^{**} \) (proof)
lemma lset-P-V [simp]: \( \text{lset } P \subseteq V \) (proof)
lemma lset-ltl-P-V [simp]: \( \text{lset } (\text{ltl } P) \subseteq V \) (proof)

lemma finite-llast-deadend [simp]: \( \text{finite } P \implies \text{deadend } (\text{llast } P) \) (proof)
lemma finite-llast-V [simp]: \( \text{finite } P \implies \text{llast } P \in V \) (proof)

If a path visits a deadend, it is winning for the other player.

lemma visits-deadend:
  assumes \( \text{lset } P \cap \text{deadends } p \neq \{\} \)
  shows \( \text{winning-path } p^{**} P \) (proof)

end

end

4 Positional Strategies

theory Strategy
imports
  Main
  ParityGame
begin

4.1 Definitions

A strategy is simply a function from nodes to nodes. We only consider positional strategies.

type-synonym \('a Strategy = 'a \Rightarrow 'a\)

A valid strategy for player \( p \) is a function assigning a successor to each node in \( \text{VV } p \).

definition (in ParityGame) strategy :: Player \Rightarrow 'a Strategy \Rightarrow bool where
  strategy p \sigma \equiv \forall v \in \text{VV } p. \neg \text{deadend } v \implies v \mapsto \sigma v

lemma (in ParityGame) strategyI [intro]:
  (\forall v. [ v \in \text{VV } p; \neg \text{deadend } v ] \implies v \mapsto \sigma v) \implies \text{strategy } p \sigma
  (proof)

4.2 Strategy-Conforming Paths

If \( \text{path-conforms-with-strategy } p \ P \ \sigma \) holds, then we call \( P \) a \( \sigma \)-path. This means that \( P \) follows \( \sigma \) on all nodes of player \( p \) except maybe the last node on the path.

coinductive (in ParityGame) path-conforms-with-strategy
  :: Player \Rightarrow 'a Path \Rightarrow 'a Strategy \Rightarrow bool where
  path-conforms-LNil: \( \text{path-conforms-with-strategy } p \ LNil \sigma \)
| path-conforms-LCons-LNil: \( \text{path-conforms-with-strategy } p \ (\text{LCons } v \ LNil) \ \sigma \)
Define a locale for valid maximal paths that conform to a given strategy, because we need this concept quite often. However, we are not yet able to add interesting lemmas to this locale. We will do this at the end of this section, where we have more lemmas available.

locale \texttt{vmc-path} = \texttt{vm-path} +
  \texttt{fixes } p \sigma \texttt{ assumes } P:\texttt{-conforms } [\texttt{simp}]: \texttt{path-conforms-with-strategy } p \ P \sigma

Similarly, define a locale for valid maximal paths that conform to given strategies for both players.

locale \texttt{vmc2-path} = \texttt{comp? : vmc-path } G \ P \ v0 \ p \star \sigma' + \texttt{vmc-path G P } v0 \ p \ \sigma

for \( G \ P \ v0 \ p \ \sigma \ \sigma' \)

4.3 An Arbitrary Strategy

context \texttt{ParityGame} begin

Define an arbitrary strategy. This is useful to define other strategies by overriding part of this strategy.

\texttt{definition } \sigma:\texttt{-arbitrary } \equiv \lambda \ v. \ \texttt{SOME } w. \ \texttt{v \rightarrow w}

\texttt{lemma valid-arbitrary-strategy } [\texttt{simp}]: \texttt{strategy } p \ \sigma:\texttt{-arbitrary } \langle \texttt{proof} \rangle

4.4 Valid Strategies

\texttt{lemma valid-strategy-updates}:[ \texttt{strategy } p \ \sigma; \ v0 \rightarrow w0 ] \implies \texttt{strategy } p (\sigma(v0 := w0))

\langle \texttt{proof} \rangle

\texttt{lemma valid-strategy-updates-set}:\texttt{ assumes } \texttt{strategy } p \ \sigma \ \land \ v. \ \texttt{v \in A; v \in VV \ p; \neg deadend v} \implies \texttt{v \rightarrow v'}

\texttt{shows } \texttt{strategy } p (\texttt{override-on } \sigma \ \sigma' \ A)

\langle \texttt{proof} \rangle

\texttt{lemma valid-strategy-updates-set-strong}:\texttt{ assumes } \texttt{strategy } p \ \sigma \ \texttt{strategy } p \ \sigma'

\texttt{shows } \texttt{strategy } p (\texttt{override-on } \sigma \ \sigma' \ A)

\langle \texttt{proof} \rangle

\texttt{lemma subgame-strategy-stays-in-subgame}:\texttt{ assumes } \sigma: \texttt{ParityGame.strategy (subgame } V') \ p \ \sigma

\texttt{and } \ v \in \texttt{ParityGame.VV (subgame } V') \ p \ \neg \texttt{Digraph.deadend (subgame } V') \ v

\texttt{shows } \sigma \ v \in V'

\langle \texttt{proof} \rangle

\texttt{lemma valid-strategy-supergame}:\texttt{ assumes } \sigma: \texttt{strategy } p \ \sigma

\texttt{and } \sigma': \texttt{ParityGame.strategy (subgame } V') \ p \ \sigma'

\texttt{and } \texttt{G'-no-deadends}: \land \ v. \ v \in V' \implies \neg \texttt{Digraph.deadend (subgame } V') \ v

\langle \texttt{proof} \rangle
shows strategy p (override-on $\sigma \sigma' V'$) (is strategy p $\sigma$)
(proof)

lemma valid-strategy-in-V: [ strategy p $\sigma$; $v \in VV p$; $\neg$deadend $v$ ] $\implies$ $\sigma v \in V$
(proof)

lemma valid-strategy-only-in-V: [ strategy p $\sigma$; $\bigwedge v. v \in V$ $\implies$ $\sigma v = \sigma' v$ ] $\implies$ strategy p $\sigma'$
(proof)

4.5 Conforming Strategies

lemma path-conforms-with-strategy-ltl [intro]:
 path-conforms-with-strategy p $P \sigma$ $\implies$ path-conforms-with-strategy p (ltl $P$) $\sigma$
(proof)

lemma path-conforms-with-strategy-drop:
 path-conforms-with-strategy p $P \sigma$ $\implies$ path-conforms-with-strategy p (ldropn n $P$) $\sigma$
(proof)

lemma path-conforms-with-strategy-prefix:
 path-conforms-with-strategy p $P \sigma$ $\implies$ lprefix $P'$ $P$ $\implies$ path-conforms-with-strategy p $P'$ $\sigma$
(proof)

lemma path-conforms-with-strategy-irrelevant:
 assumes path-conforms-with-strategy p $P \sigma v \notin lset P$
 shows path-conforms-with-strategy p $P (\sigma(v := w))$
(proof)

lemma path-conforms-with-strategy-irrelevant-deadend:
 assumes path-conforms-with-strategy p $P \sigma$ $\neg$deadend $v \vee v \notin VV p$ valid-path $P$
 shows path-conforms-with-strategy p $P (\sigma(v := w))$
(proof)

lemma path-conforms-with-strategy-irrelevant-updates:
 assumes path-conforms-with-strategy p $P \sigma \bigwedge v. v \in lset P$ $\implies$ $\sigma v = \sigma' v$
 shows path-conforms-with-strategy p $P \sigma'$
(proof)

lemma path-conforms-with-strategy-irrelevant':
 assumes path-conforms-with-strategy p $P (\sigma(v := w)) v \notin lset P$
 shows path-conforms-with-strategy p $P \sigma$
(proof)

lemma path-conforms-with-strategy-irrelevant-deadend':
 assumes path-conforms-with-strategy p $P (\sigma(v := w))$ deadend $v \vee v \notin VV p$ valid-path $P$
 shows path-conforms-with-strategy p $P \sigma$
(proof)

lemma path-conforms-with-strategy-start:
 path-conforms-with-strategy p (LCons v (LCons w $P$)) $\sigma$ $\implies$ $v \in VV p$ $\implies$ $\sigma v = w$
(proof)
Lemma path-conforms-with-strategy-lappend:

**Assumes**
- \( P : \text{finite} P \lor \text{null} P \) path-conforms-with-strategy \( p \ P \sigma \)
- and \( \neg \text{null} P \) path-conforms-with-strategy \( p \ P' \sigma \)
- and conforms: \( \text{llast} P \in VV p \implies (\text{llast} P) = \text{lhbp} P' \)

**Shows** path-conforms-with-strategy \( p \) (lappend \( P \ P' \)) \( \sigma \)

(Proof)

Lemma path-conforms-with-strategy-VVpstar:

**Assumes** \( \text{lset} P \subseteq VV p^{**} \)

**Shows** path-conforms-with-strategy \( p \ P \sigma \)

(Proof)

Lemma subgame-path-conforms-with-strategy:

**Assumes** \( V' : V' \subseteq V \) and \( P : \text{path-conforms-with-strategy} \ p \ P \sigma \ \text{lset} P \subseteq V' \)

**Shows** ParityGame.path-conforms-with-strategy (subgame \( V' \)) \( p \ P \sigma \)

(Proof)

Lemma (in vmc-path) subgame-path-vmc-path:

**Assumes** \( V' : V' \subseteq V \) and \( P : \text{lset} P \subseteq V' \)

**Shows** vmc-path (subgame \( V' \)) \( p v0 p \sigma \)

(Proof)

### 4.6 Greedy Conforming Path

Given a starting point and two strategies, there exists a path conforming to both strategies. Here we define this path. Incidentally, this also shows that the assumptions of the locales vmc-path and vmc2-path are satisfiable.

We are only interested in proving the existence of such a path, so the definition (i.e., the implementation) and most lemmas are private.

Context begin

Private primcorec greedy-conforming-path :: Player \to 'a Strategy \to 'a Strategy \to 'a \to 'a Path

where

\[
\text{greedy-conforming-path} \ p \sigma \sigma' v0 = \\
\text{LCons} v0 \ (\text{if} \ \text{deadend} v0) \\
\text{then} L\text{Nil} \\
\text{else} \text{if} \ v0 \in VV p \\
\text{then} \text{greedy-conforming-path} \ p \sigma \sigma' (v0) \\
\text{else} \text{greedy-conforming-path} \ p \sigma \sigma' (v0))
\]

Private lemma greedy-path-LNil: greedy-conforming-path \( p \sigma \sigma' v0 \neq L\text{Nil} \)

(Proof) Lemma greedy-path-lhd: greedy-conforming-path \( p \sigma \sigma' v0 = L\text{Cons} v P \implies v = v0 \)

(Proof) Lemma greedy-path-deadend-v0: greedy-conforming-path \( p \sigma \sigma' v0 = L\text{Cons} v P \implies P = L\text{Nil} \leftrightarrow \text{deadend} v0 \)

(Proof) Corollary greedy-path-deadend-v:

\[
\text{greedy-conforming-path} \ p \sigma \sigma' v0 = L\text{Cons} v P \implies P = L\text{Nil} \leftrightarrow \text{deadend} v
\]

(Proof) Corollary greedy-path-deadend-v': greedy-conforming-path \( p \sigma \sigma' v0 = L\text{Cons} v L\text{Nil} \implies \text{deadend} v

17
\langle proof \rangle \textbf{lemma} \text{greedy-path-ltl:}
\begin{align*}
\text{assumes} & \text{ greedy-conforming-path } p \sigma \sigma' v_0 = LCons v P \\
\text{shows} & \quad P = LNil \lor P = \text{greedy-conforming-path } p \sigma \sigma' (\sigma v_0) \lor P = \text{greedy-conforming-path } p \sigma' (\sigma' v_0)
\end{align*}
\langle proof \rangle

\textbf{lemma} \text{greedy-path-ltl-ex:}
\begin{align*}
\text{assumes} & \text{ greedy-conforming-path } p \sigma \sigma' v_0 = LCons v P \\
\text{shows} & \quad P = LNil \lor (\exists v. P = \text{greedy-conforming-path } p \sigma \sigma' v) \\
\text{assumes} & \text{ greedy-conforming-path } p \sigma \sigma' v_0 = LCons v P v_0 \in VV p \neg \text{deadend } v_0 \\
\text{shows} & \quad \sigma v_0 = lhd P \\
\text{assumes} & \text{ greedy-conforming-path } p \sigma \sigma' v_0 = LCons v P v_0 \in VV p^{**} \neg \text{deadend } v_0 \\
\text{shows} & \quad \sigma' v_0 = lhd P
\end{align*}
\langle proof \rangle

\textbf{lemma} \text{greedy-path-ltl-VVp:}
\begin{align*}
\text{assumes} & \text{ greedy-conforming-path } p \sigma \sigma' v_0 = LCons v0 P v_0 \in VV p \neg \text{deadend } v_0 \\
\text{shows} & \quad \sigma v_0 = \text{lhd } P \\
\text{assumes} & \text{ greedy-conforming-path } p \sigma \sigma' v_0 = LCons v P v_0 \in VV p^{**} \neg \text{deadend } v_0 \\
\text{shows} & \quad \sigma' v_0 = \text{lhd } P
\end{align*}
\langle proof \rangle

\textbf{lemma} \text{greedy-path-ltl-VVpstar:}
\begin{align*}
\text{assumes} & \text{ greedy-conforming-path } p \sigma \sigma' v_0 = LCons v0 P v_0 \in VV p^{**} \neg \text{deadend } v_0 \\
\text{shows} & \quad \sigma' v_0 = \text{lhd } P
\end{align*}
\langle proof \rangle

\textbf{lemma} \text{greedy-path-ltl-VVp:}
\begin{align*}
\text{assumes} & \text{ greedy-conforming-path } p \sigma \sigma' v_0 = LCons v0 P v_0 \in VV p^{**} \neg \text{deadend } v_0 \\
\text{shows} & \quad \sigma v_0 = \text{lhd } P \\
\text{assumes} & \text{ greedy-conforming-path } p \sigma \sigma' v_0 = LCons v P v_0 \in VV p^{**} \neg \text{deadend } v_0 \\
\text{shows} & \quad \sigma' v_0 = \text{lhd } P
\end{align*}
\langle proof \rangle

\textbf{lemma} \text{strategy-conforming-path-exists:}
\begin{align*}
\text{assumes} & \quad v_0 \in V \text{ strategy } p \sigma \sigma' \sigma'' \\
\text{obtains} & \quad P \text{ where } \text{vmc2-path } G P v_0 p \sigma \sigma' \\
\text{obtains} & \quad P \text{ where } \text{vmc-path } G P v_0 p \sigma \sigma'
\end{align*}
\langle proof \rangle

\textbf{lemma} \text{strategy-conforming-path-exists-single:}
\begin{align*}
\text{assumes} & \quad v_0 \in V \text{ strategy } p \sigma \sigma'
\text{obtains} & \quad P \text{ where } \text{vmc-path } G P v_0 p \sigma \sigma'
\end{align*}
\langle proof \rangle

end

end

\section{4.7 Valid Maximal Conforming Paths}

Now is the time to add some lemmas to the locale \textit{vmc-path}.

\textbf{context} \textit{vmc-path} \textbf{begin}
\textbf{lemma} \textit{Ptl-conforms} \textbf{[simp]}: path-conforms-with-strategy p (ltl P) \sigma
\langle proof \rangle
\textbf{lemma} \textit{Pdrop-conforms} \textbf{[simp]}: path-conforms-with-strategy p (ldropn n P) \sigma
\langle proof \rangle
\textbf{lemma} \textit{prefix-conforms} \textbf{[simp]}: path-conforms-with-strategy p (ltake n P) \sigma
\langle proof \rangle
\textbf{lemma} \textit{extension-conforms} \textbf{[simp]}:
\begin{align*}
\langle v' \in VV p \implies \sigma v' = v_0 \rangle & \implies \text{path-conforms-with-strategy p } (LCons v' P) \sigma
\end{align*}
\langle proof \rangle
end

18
4.8 Valid Maximal Conforming Paths with One Edge

We define a locale for valid maximal conforming paths that contain at least one edge. This is equivalent to the first node being no deadend. This assumption allows us to prove much stronger lemmas about \( \text{ltl} P \) compared to \( \text{vmc-path} \).

locale \( \text{vmc-path-no-deadend} = \text{vmc-path} + \)

assumes \( v0\text{-no-deadend} \) [simp]: \text{\( \neg \)deadend} \( v0 \)

begin

definition \( w0 \equiv \text{lhd} (\text{ltl} \ p) \)

lemma \( \text{Ptl-not-null} \) [simp]: \text{\( \neg \)lnull} \( \text{ltl} \ \ p \)

lemma \( \text{Ptl-LCons} \): \( \text{ltl} \ \ p = \text{LCons} \ w0 \ (\text{ltl} \ (\text{ltl} \ \ p)) \) (proof)

lemma \( \text{P-LCons}': \ p = \text{LCons} \ w0 \ (\text{LCons} \ w0 \ (\text{ltl} \ (\text{ltl} \ \ p))) \) (proof)

lemma \( v0\text{-edge-w0} \) [simp]: \( v0 \text{\( \rightarrow \)} w0 \) (proof)

lemma \( \text{Ptl-0} \): \( \text{ltl} \ \ p \ \ \text{Suc} \ 0 = \text{lhd} (\text{ltl} \ \ p) \) (proof)

lemma \( \text{P-Suc-0} \): \( \text{P} \ \text{Suc} \ 0 = w0 \) (proof)

lemma \( P\text{-tl-edge} \) [simp]: \( v0 \text{\( \rightarrow \)} \text{lhd} (\text{ltl} \ \ p) \) (proof)

lemma \( v0\text{-conforms} \): \( v0 \in \text{VV} \ \ p \implies \ \sigma \ v0 = w0 \) (proof)

lemma \( w0\text{-V} \) [simp]: \( w0 \in \text{V} \) (proof)

lemma \( w0\text{-lset-P} \) [simp]: \( w0 \in \text{lset} \ \ p \) (proof)

lemma \( \text{vmc-path-ltl} \) [simp]: \( \text{vmc-path} \ \text{G} \ (\text{ltl} \ \ p) \ \text{w0} \ \ p \ \ \sigma \) (proof)

end
context vmc-path begin

lemma vmc-path-lnull-ltl-no-deadend:
\[ \lnot \lnull \langle \ltl P \rangle \Rightarrow \text{vmc-path-no-deadend} G P v0 p \sigma \]
(proof)

lemma vmc-path-conforms:
assumes enat \((\Suc n)\) < llength P $ n \in VV p
shows \(\sigma (P $ n) = P $ \Suc n\)
(proof)

4.9 lset Induction Schemas for Paths

Let us define an induction schema useful for proving \(\text{lset} P \subseteq S\).

lemma vmc-path-lset-induction [consumes 1, case-names base step]:
assumes Q P
and base: v0 \in S
and step-assumption: \(\land P v0. \, [ \text{vmc-path-no-deadend} G P v0 p \sigma; v0 \in S; Q P ] \Rightarrow Q \langle \ltl P \rangle \land (\text{vmc-path-no-deadend}.w0 P) \in S\)
shows lset P \subseteq S
(proof)

\[ ?Q P; v0 \in ?S; \land P v0. \, [\text{vmc-path-no-deadend} G P v0 p \sigma; v0 \in ?S; ?Q P] \Rightarrow ?Q \langle \ltl P \rangle \land \text{vmc-path-no-deadend}.w0 P \in ?S \Rightarrow \text{lset} P \subseteq ?S \text{ without the Q predicate.} \]

corollary vmc-path-lset-induction-simple [case-names base step]:
assumes base: v0 \in S
and step: \(\land P v0. \, [\text{vmc-path-no-deadend} G P v0 p \sigma; v0 \in S ] \Rightarrow \text{vmc-path-no-deadend}.w0 P \in S\)
shows lset P \subseteq S
(proof)

Another induction schema for proving \(\text{lset} P \subseteq S\) based on closure properties.

lemma vmc-path-lset-induction-closed-subset [case-names VVp VVpstar v0 disjoint]:
assumes VVp: \(\land v. \, [v \in S; \lnot \text{deadend} v; v \in VV p ] \Rightarrow \sigma v \in S \cup T\)
and VVpstar: \(\land v w. \, [v \in S; \lnot \text{deadend} v; v \in VV p**; v \rightarrow w ] \Rightarrow w \in S \cup T\)
and v0: v0 \in S
and disjoint: lset P \cap T = \{\}
shows lset P \subseteq T
(proof)

end

end

5 Attracting Strategies

theory AttractingStrategy
imports
  Main
  Strategy
Here we introduce the concept of attracting strategies.

context ParityGame begin

5.1 Paths Visiting a Set

A path that stays in $A$ until eventually it visits $W$.

definition visits-via $P$ $A$ $W$ $\equiv \exists n. \text{enat } n < \text{length } P \land P \downarrow n \in W \land \text{lset } (\text{ltake } (\text{enat } n) P) \subseteq A$

lemma visits-via-monotone: $[\text{visits-via } P A W; A \subseteq A'] \implies \text{visits-via } P A' W$

lemma visits-via-visits: $\text{visits-via } P A W \implies \text{lset } P \cap W \neq \{\}$

lemma (in vmc-path) visits-via-trivial: $v0 \in W \implies \text{visits-via } P A W$

lemma visits-via-LCons: 
  assumes visits-via $P$ $A$ $W$
  shows visits-via $(\text{LCons } v0 P) (\text{insert } v0 A) W$

lemma (in vmc-path-no-deadend) visits-via-ltl: 
  assumes visits-via $P$ $A$ $W$
  and $v0: v0 \notin W$
  shows visits-via $\text{ltl } P A W$

lemma (in vm-path) visits-via-deadend: 
  assumes visits-via $P$ $A$ $(\text{deadends } p)$
  shows winning-path $p^* P$

5.2 Attracting Strategy from a Single Node

All $\sigma$-paths starting from $v0$ visit $W$ and until then they stay in $A$.

definition strategy-attracts-via :: Player $\Rightarrow$ 'a Strategy $\Rightarrow$ 'a $\Rightarrow$ 'a set $\Rightarrow$ 'a set $\Rightarrow$ bool where
  strategy-attracts-via $p$ $\sigma$ $v0$ $A$ $W$ $\equiv \forall P. \text{vmc-path } G P v0 p \sigma \implies \text{visits-via } P A W$

lemma (in vmc-path) strategy-attracts-viaE: 
  assumes strategy-attracts-via $p$ $\sigma$ $v0$ $A$ $W$
  shows visits-via $P A W$

lemma (in vmc-path) strategy-attracts-via-SucE: 
  assumes strategy-attracts-via $p$ $\sigma$ $v0$ $A$ $W$
  shows $\exists n. \text{enat } (\text{Suc } n) < \text{length } P \land P \downarrow \text{Suc } n \in W \land \text{lset } (\text{ltake } (\text{enat } (\text{Suc } n)) P) \subseteq A$

21
lemma (in vmc-path) strategy-attracts-via-lset:
  assumes strategy-attracts-via p σ v0 A W
  shows lset P ∩ W ≠ { }
⟨proof⟩

lemma strategy-attracts-via-v0:
  assumes σ: strategy p σ strategy-attracts-via p σ v0 A W
  and v0: v0 ∈ V
  shows v0 ∈ A ∪ W
⟨proof⟩
corollary strategy-attracts-not-outside:
  [ v0 ∈ V − A − W; strategy p σ ] ⇒ ¬ strategy-attracts-via p σ v0 A W
⟨proof⟩

lemma strategy-attracts-via-intro:
  assumes P, vmc-path G P v0 p σ ⟹ visits-via P A W
  shows strategy-attracts-via p σ v0 A W
⟨proof⟩

lemma strategy-attracts-via-no-deadends:
  assumes v ∈ V v ∈ A − W strategy-attracts-via p σ v A W
  shows ¬ deadend v
⟨proof⟩

lemma attractor-strategy-on-extends:
  [ strategy-attracts-via p σ v0 A W; A ⊆ A’ ] ⇒ strategy-attracts-via p σ v0 A’ W
⟨proof⟩

lemma strategy-attracts-via-trivial:
  v0 ∈ W ⇒ strategy-attracts-via p σ v0 A W
⟨proof⟩

lemma strategy-attracts-via-successor:
  assumes σ: strategy p σ strategy-attracts-via p σ v0 A W
  and v0: v0 ∈ A − W
  and w0: v0 → w0 v0 ∈ VV p ⟹ σ v0 = w0
  shows strategy-attracts-via p σ w0 A W
⟨proof⟩

lemma strategy-attracts-VVp:
  assumes σ: strategy p σ strategy-attracts-via p σ v0 A W
  and v: v0 ∈ A − W v0 ∈ VV p ¬ deadend v0
  shows σ v0 ∈ A ∪ W
⟨proof⟩

lemma strategy-attracts-VVpistar:
  assumes strategy p σ strategy-attracts-via p σ v0 A W
  and v0 ∈ A − W v0 / ∈ VV p w0 ∈ V − A − W
  shows ¬ v0 → w0
⟨proof⟩
5.3 Attracting strategy from a set of nodes

All σ-paths starting from A visit W and until then they stay in A.

definition strategy-attracts :: Player ⇒ 'a Strategy ⇒ 'a set ⇒ 'a set ⇒ bool where
strategy-attracts p σ A W ≡ ∀ v0 ∈ A. strategy-attracts-via p σ v0 A W

lemma (in vmc-path) strategy-attractsE:
  assumes strategy-attracts p σ A W v0 ∈ A
  shows visits-via P A W
  ⟨proof⟩

lemma strategy-attractsI [intro]:
  assumes ∃ P v. [ v ∈ A; vmc-path G P v p σ ] ⇒ visits-via P A W
  shows strategy-attracts p σ A W
  ⟨proof⟩

lemma (in vmc-path) strategy-attracts-lset:
  assumes strategy-attracts p σ A W v0 ∈ A
  shows lset P ∩ W ≠ {}
  ⟨proof⟩

lemma strategy-attracts-empty [simp]: strategy-attracts p σ {} W ⟨proof⟩

lemma strategy-attracts-invalid-path:
  assumes P: P = LCons v (LCons w P') v ∈ A − W w /∈ A ∪ W
  shows ¬visits-via P A W (is ¬?A)
  ⟨proof⟩

If A is an attractor set of W and an edge leaves A without going through W, then v belongs to VV p and the attractor strategy σ avoids this edge. All other cases give a contradiction.

lemma strategy-attracts-does-not-leave:
  assumes σ: strategy-attracts p σ A W strategy p σ
  and v: v→w v ∈ A − W w /∈ A ∪ W
  shows v ∈ VV p ∧ σ v ≠ w
  ⟨proof⟩

Given an attracting strategy σ, we can turn every strategy σ' into an attracting strategy by overriding σ' on a suitable subset of the nodes. This also means that an attracting strategy is still attracting if we override it outside of A − W.

lemma strategy-attracts-irrelevant-override:
  assumes strategy-attracts p σ A W strategy p σ strategy p σ'
  shows strategy-attracts p (override-on σ' σ (A − W)) A W
  ⟨proof⟩

lemma strategy-attracts-trivial [simp]: strategy-attracts p σ W W ⟨proof⟩

If a σ-conforming path P hits an attractor A, it will visit W.

lemma (in vmc-path) attracted-path:
  assumes W ⊆ V

23
theory Attractor
imports 
  Main  
  AttractingStrategy
begin 

Here we define the $p$-attractor of a set of nodes.

context ParityGame begin

We define the conditions for a node to be directly attracted from a given set.

definition directly-attracted :: Player $\Rightarrow$ 'a set $\Rightarrow$ 'a set where
  directly-attracted $p$ $S \equiv \{ v \in V - S . \neg$deadend $v \wedge$
  $\{ v \in V V p \rightarrow (\exists w . v \rightarrow w \wedge w \in S) \}$
  $\wedge \{ v \in V V p^{**} \rightarrow (\forall w . v \rightarrow w \rightarrow w \in S) \}\}$

abbreviation attractor-step $p$ $W$ $S \equiv W \cup S \cup$ directly-attracted $p$ $S$

The $p$-attractor set of $W$, defined as a least fixed point.

definition attractor :: Player $\Rightarrow$ 'a set $\Rightarrow$ 'a set where
  attractor $p$ $W \equiv \text{lfp } (\text{attractor-step } p \ W)$

6.1 directly-attracted

Show a few basic properties of directly-attracted.

lemma directly-attracted-disjoint $[\text{simp}]$: directly-attracted $p$ $W \cap W = \{\}$
and directly-attracted-empty $[\text{simp}]$: directly-attracted $p \ \{\} = \{\}$
and directly-attracted-V-empty \[\text{simp}]: \text{directly-attracted } p \ V = \{\}
and directly-attracted-bounded-by-V \[\text{simp}]: \text{directly-attracted } p \ W \subseteq V
and directly-attracted-contains-no-deadends \[\text{elim}]: v \in \text{directly-attracted } p \ W \implies \neg \text{deadend } v
\langle \text{proof} \rangle

6.2 attractor-step

\textbf{lemma attractor-step-empty:} \text{attractor-step } p \ \{\} \ \{\} = \{
and \text{attractor-step-bounded-by-V:} \ [ W \subseteq V; S \subseteq V ] \implies \text{attractor-step } p \ W \ S \subseteq V
\langle \text{proof} \rangle

The definition of attractor uses \text{lfp}. For this to be well-defined, we need show that \text{attractor-step} is monotone.

\textbf{lemma attractor-step-mono:} \text{mono}(\text{attractor-step } W)
\langle \text{proof} \rangle

6.3 Basic Properties of an Attractor

\textbf{lemma attractor-unfolding:} \text{attractor } p \ W = \text{attractor-step } p \ W \ (\text{attractor } p \ W)
\langle \text{proof} \rangle
\textbf{lemma attractor-lowerbound:} \text{attractor-step } p \ W \ S \subseteq S \implies \text{attractor } p \ W \subseteq S
\langle \text{proof} \rangle
\textbf{lemma attractor-set-non-empty:} W \neq \{\} \implies \text{attractor } p \ W \neq \{
and \text{attractor-set-base:} W \subseteq \text{attractor } p \ W
\langle \text{proof} \rangle
\textbf{lemma attractor-in-V:} W \subseteq V \implies \text{attractor } p \ W \subseteq V
\langle \text{proof} \rangle

6.4 Attractor Set Extensions

\textbf{lemma attractor-set-VVp:}
\text{assumes} v \in VV \ p v \rightarrow w \ w \in \text{attractor } p \ W
\text{shows} v \in \text{attractor } p \ W
\langle \text{proof} \rangle
\textbf{lemma attractor-set-VVpstar:}
\text{assumes} \neg \text{deadend } v \ \land \ w. v \rightarrow w \implies w \in \text{attractor } p \ W
\text{shows} v \in \text{attractor } p \ W
\langle \text{proof} \rangle

6.5 Removing an Attractor

\textbf{lemma removing-attractor-induces-no-deadends:}
\text{assumes} v \in S - \text{attractor } p \ W \ v \rightarrow w \ w \in S \land w. [ v \in VV \ p**; v \rightarrow w ] \implies w \in S
\text{shows} \exists w \in S - \text{attractor } p \ W. v \rightarrow w
\langle \text{proof} \rangle

Removing the attractor sets of deadends leaves a subgame without deadends.

\textbf{lemma subgame-without-deadends:}
\text{assumes} V' - \text{def}: V' = V - \text{attractor } p \ (\text{deadends } p**) - \text{attractor } p** \ (\text{deadends } p****)
\text{(is } V' = V - \ ?A - \ ?B)\text{ and } v: v \in V'_{\text{subgame } V'}
6.6 Attractor Set Induction

lemma mono-restriction-is-mono: mono \( f \) \( \Rightarrow \) mono \( (\lambda S. f (S \cap V)) \)

(\langle proof \rangle)

Here we prove a powerful induction schema for attractor. Being able to prove this is the only reason why we do not use \texttt{inductive\_set} to define the attractor set.

See also https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2015-October/msg00123.html

lemma attractor-set-induction [consumes 1, case-names step union]:
  assumes \( W \subseteq V \)
  and step: \( \forall S. S \subseteq V \Rightarrow P S \Rightarrow P (\text{attractor-step} p W S) \)
  and union: \( \forall M. \forall S \in M. S \subseteq V \land P S \Rightarrow P (\bigcup M) \)
  shows \( P (\text{attractor} p W) \)

(\langle proof \rangle)

There cannot exist winning strategies for both players for the same node.

lemma winning-strategy-only-for-one-player:

There cannot exist winning strategies for both players for the same node.
assumes $\sigma$: strategy $p \sigma$ winning-strategy $p \sigma v$
and $\sigma':$ strategy $p** \sigma' v$
and $v: v \in V$
shows False
(proof)

7.1 Deadends

lemma no-winning-strategy-on-deadends:
assumes $v \in V V p$ deadend $v$ strategy $p \sigma$
shows $\neg$winning-strategy $p \sigma v$
(proof)

lemma winning-strategy-on-deadends:
assumes $v \in V V p$ deadend $v$ strategy $p \sigma$
shows winning-strategy $p** \sigma v$
(proof)

7.2 Extension Theorems

lemma strategy-extends-$V V p$:
assumes $v0: v0 \in V V p$ $v0$ deadend $v0$
and $\sigma$: strategy $p \sigma$ winning-strategy $p \sigma v0$
shows winning-strategy $p \sigma (\sigma v0)$
(proof)

lemma strategy-extends-$V V p$star:
assumes $v0: v0 \in V V p** v0$ $\rightarrow$ $w0$
and $\sigma$: winning-strategy $p \sigma v0$
shows winning-strategy $p \sigma w0$
(proof)

lemma strategy-extends-backwards-$V V p$star:
assumes $v0: v0 \in V V p**$
and $\sigma$: strategy $p \sigma \land w. v0 \rightarrow w$ $\Rightarrow$ winning-strategy $p \sigma w$
shows winning-strategy $p \sigma v0$
(proof)

lemma strategy-extends-backwards-$V V p$:
assumes $v0: v0 \in V V p \sigma v0 = w v0 \rightarrow w$
and $\sigma$: strategy $p \sigma$ winning-strategy $p \sigma w$
shows winning-strategy $p \sigma v0$
(proof)

end — context ParityGame

end

8 Well-Ordered Strategy

theory WellOrderedStrategy
Constructing a uniform strategy from a set of strategies on a set of nodes often works by well-ordering the strategies and then choosing the minimal strategy on each node. Then every path eventually follows one strategy because we choose the strategies along the path to be non-increasing in the well-ordering.

The following locale formalizes this idea.

We will use this to construct uniform attractor and winning strategies.

locale WellOrderedStrategies = ParityGame +
  fixes S :: `'a set`
  and p :: `Player`
  — The set of good strategies on a node v
  and good :: `'a ⇒ 'a Strategy set`
  and r :: `(Strategy × 'a Strategy) set`
  assumes S-V: `S ⊆ V`
  — r is a wellorder on the set of all strategies which are good somewhere.
  and r-wo: `well-order-on {∃ v ∈ S. σ ∈ good v} r`
  — Every node has a good strategy.
  and good-ex: `∀ v ∈ S. σ ∈ good v`
  — good strategies are well-formed strategies.
  and good-strategies: `∀ v ∈ S. σ ∈ good v`
  — A good strategy on v is also good on possible successors of v.
  and strategies-continue: `∀ v w σ. [ v ∈ S; v → w; v ∈ V V p ⇒ σ v = w; σ ∈ good v ] ⇒ σ ∈ good w`

begin

The set of all strategies which are good somewhere.

abbreviation Strategies ≡ `{∃ v ∈ S. σ ∈ good v}`

definition minimal-good-strategy where
  minimal-good-strategy v σ ≡ σ ∈ good v ∧ (∀ σ'. (σ', σ) ∈ r − Id ⇒ σ' /∈ good v)

no-notation binomial (infixl `choose 65`)

Among the good strategies on v, choose the minimum.

definition choose where
  choose v ≡ THE σ. minimal-good-strategy v σ

Define a strategy which uses the minimum strategy on all nodes of S. Of course, we need to prove that this is a well-formed strategy.

definition well-ordered-strategy where
  well-ordered-strategy ≡ override-on σ-arbitrary (λ v. choose v v) S

Show some simple properties of the binary relation r on the set Strategies.

lemma r-refl [simp]: refl-on Strategies r
  ⟨proof⟩
**lemma** \( r\text{-}total \) \([\text{simp}]: \text{total-on } \text{Strategies } r \)

\(\langle \text{proof} \rangle\)

**lemma** \( r\text{-}trans \) \([\text{simp}]: \text{trans } r \)

\(\langle \text{proof} \rangle\)

**lemma** \( r\text{-}wf \) \([\text{simp}]: \text{wf } (r - \text{Id}) \)

\(\langle \text{proof} \rangle\)

\(\text{choose}\) always chooses a minimal good strategy on \( S \).

**lemma** choose-works:
- \( \text{assumes } v \in S \)
- \( \text{shows } \text{minimal-good-strategy } v \) (choose \( v \))

\(\langle \text{proof} \rangle\)

**corollary**
- \( \text{assumes } v \in S \)
- \( \text{shows } \text{choose-good}: \text{choose } v \in \text{good } v \)

\(\langle \text{proof} \rangle\)

**corollary** choose-in-Strategies: \( v \in S \implies \text{choose } v \in \text{Strategies} \)

\(\langle \text{proof} \rangle\)

**lemma** well-ordered-strategy-valid: \( \text{strategy } p \ \text{well-ordered-strategy} \)

\(\langle \text{proof} \rangle\)

### 8.1 Strategies on a Path

Maps a path to its strategies.

**definition** \( \text{path-strategies} \equiv \text{lmap choose} \)

**lemma** path-strategies-in-Strategies:
- \( \text{assumes } \text{lset } P \subseteq S \)
- \( \text{shows } \text{lset } (\text{path-strategies } P) \subseteq \text{Strategies} \)

\(\langle \text{proof} \rangle\)

**lemma** path-strategies-good:
- \( \text{assumes } \text{lset } P \subseteq S \ \text{enat } n < \text{llength } P \)
- \( \text{shows } \text{path-strategies } P \$(n) \in \text{good } (P \$ n) \)

\(\langle \text{proof} \rangle\)

**lemma** path-strategies-strategy:
- \( \text{assumes } \text{lset } P \subseteq S \ \text{enat } n < \text{llength } P \)
- \( \text{shows } \text{strategy } p (\text{path-strategies } P \$(n)) \)

\(\langle \text{proof} \rangle\)

**lemma** path-strategies-monotone-Suc:
- \( \text{assumes } P: \text{lset } P \subseteq S \ \text{valid-path } P \ \text{path-conforms-with-strategy } p P \ \text{well-ordered-strategy} \)

\( \text{enat } (\text{Suc } n) < \text{llength } P \)
- \( \text{shows } (\text{path-strategies } P \$(\text{Suc } n), \text{path-strategies } P \$ n) \in r \)

\(\langle \text{proof} \rangle\)
lemma path-strategies-monotone:
  assumes \( P : lset P \subseteq S \) valid-path \( P \) path-conforms-with-strategy \( p \) \( P \) well-ordered-strategy
  \( n < m \) enat \( m < \text{llength} P \)
  shows (path-strategies \( P \) \$ \( m \), path-strategies \( P \) \$ \( n \)) \( \in r \)
  ⟨proof⟩

lemma path-strategies-eventually-constant:
  assumes \( \neg lfinite P \) lset \( P \subseteq S \) valid-path \( P \) path-conforms-with-strategy \( p \) \( P \) well-ordered-strategy
  shows \( \exists n. \forall m \geq n. \) path-strategies \( P \) \$ \( n \) = path-strategies \( P \) \$ \( m \)
  ⟨proof⟩

8.2 Eventually One Strategy

The key lemma: Every path that stays in \( S \) and follows well-ordered-strategy eventually follows one strategy because the strategies are well-ordered and non-increasing along the path.

lemma path-eventually-conforms-to-σ-map-n:
  assumes lset \( P \subseteq S \) valid-path \( P \) path-conforms-with-strategy \( p \) \( P \) well-ordered-strategy
  shows \( \exists n. \) path-conforms-with-strategy \( p \) (ldropn \( n \) \( P \)) (path-strategies \( P \) \$ \( n \))
  ⟨proof⟩

9 Winning Regions

theory WinningRegion
imports
  Main
  WinningStrategy
begin

Here we define winning regions of parity games. The winning region for player \( p \) is the set of nodes from which \( p \) has a positional winning strategy.

context ParityGame begin

definition winning-region \( p \) \equiv \{ v \in V. \exists \sigma. \text{strategy} \( p \) \( \sigma \) \& winning-strategy \( p \) \( \sigma \) \( v \) \}

lemma winning-regionI [intro]:
  assumes \( v \in V \) \( \text{strategy} \( p \) \( \sigma \) \) winning-strategy \( p \) \( \sigma \) \( v \)
  shows \( v \in \text{winning-region} \( p \) \)
  ⟨proof⟩

lemma winning-region-in-V [simp]: winning-region \( p \subseteq V \) ⟨proof⟩

lemma winning-region-deadends:
  assumes \( v \in V V \) \( p \) deadend \( v \)
  shows \( v \in \text{winning-region} \( p \)\)
9.1 Paths in Winning Regions

lemma (in vmc-path) paths-stay-in-winning-region:
assumes σ': strategy p σ' winning-strategy p σ' v0
and σ: ∨ v. v ∈ winning-region p → σ' v = σ v
shows lset P ⊆ winning-region p
(proof)

lemma (in vmc-path) path-hits-winning-region-is-winning:
assumes σ': strategy p σ' ∨ v. v ∈ winning-region p → winning-strategy p σ' v
and σ: ∨ v. v ∈ winning-region p → σ' v = σ v
and P: lset P ∩ winning-region p ≠ {} 
shows winning-path p P
(proof)

9.2 Irrelevant Updates

Updating a winning strategy outside of the winning region is irrelevant.

lemma winning-strategy-updates:
assumes σ: strategy p σ winning-strategy p σ v0
and v: v /∈ winning-region p v→w 
shows winning-strategy p (σ(v := w)) v0
(proof)

9.3 Extending Winning Regions

lemma winning-region-extends-VVp:
assumes v: v ∈ VV p v→w and w: w ∈ winning-region p 
shows v ∈ winning-region p
(proof)

Unfortunately, we cannot prove the corresponding theorem winning-region-extends-VVp* for VV p**-nodes yet. First, we need to show that there exists a uniform winning strategy on winning-region p. We will prove winning-region-extends-VVp* as soon as we have this.
end — context ParityGame

end

10 Uniform Strategies

Theorems about how to get a uniform strategy given strategies for each node.

theory UniformStrategy
imports 
  Main 
  AttractingStrategy WinningStrategy WellOrderedStrategy WinningRegion 
begin 
context ParityGame begin
10.1 A Uniform Attractor Strategy

**lemma** merge-attractor-strategies:
  **assumes** $S \subseteq V$
  **and** strategies-ex: $\forall v. v \in S \implies \exists \sigma. \text{strategy } p \sigma \land \text{strategy-attracts-via } p \sigma v S W$
  **shows** $\exists \sigma. \text{strategy } p \sigma \land \text{strategy-attracts } p \sigma S W$
  (**proof**)

10.2 A Uniform Winning Strategy

Let $S$ be the winning region of player $p$. Then there exists a uniform winning strategy on $S$.

**lemma** merge-winning-strategies:
  **shows** $\exists \sigma. \text{strategy } p \sigma \land (\forall v \in \text{winning-region } p. \text{winning-strategy } p \sigma v)$
  (**proof**)

10.3 Extending Winning Regions

Now we are finally able to prove the complement of winning-region-extends-$VVp$ for $VV p** nodes, which was still missing.

**lemma** winning-region-extends-$VVpstar$:
  **assumes** $v: v \in VV p** and $w: w \ni v \implies w \in \text{winning-region } p$
  **shows** $v \in \text{winning-region } p$
  (**proof**)

It immediately follows that removing a winning region cannot create new deadends.

**lemma** removing-winning-region-induces-no-deadends:
  **assumes** $v \in V \setminus \text{winning-region } p \land \neg \text{deadend } v$
  **shows** $\exists w \in V \setminus \text{winning-region } p. v \rightarrow w$
  (**proof**)

end — context ParityGame

end

11 Attractor Strategies

**theory** AttractorStrategy
**imports**
  Main
  Attractor UniformStrategy
**begin**

This section proves that every attractor set has an attractor strategy.

**context** ParityGame **begin**

**lemma** strategy-attracts-extends-$VVp$:
  **assumes** $\sigma: \text{strategy } p \sigma \land \text{strategy-attracts } p \sigma S W$
  **and** $v0: v0 \in VV p \land v0 \in \text{directly-attracted } p S v0 \notin S$
  **shows** $\exists \sigma. \text{strategy } p \sigma \land \text{strategy-attracts-via } p \sigma v0 (\text{insert } v0 S) W$
  (**proof**)

end
lemma strategy-attracts-extends-VVpstar:
  assumes σ: strategy-attracts p σ S W
  and v0: v0 ∉ VV p v0 ∈ directly-attracted p S
  shows strategy-attracts-via p σ v0 (insert v0 S) W
⟨proof⟩

lemma attractor-has-strategy-single:
  assumes W ⊆ V
  and v0-def: v0 ∈ attractor p W (is - ∈ ?A)
  shows ∃ σ. strategy p σ ∧ strategy-attracts-via p σ v0 ?A W
⟨proof⟩

11.1 Existence

Prove that every attractor set has an attractor strategy.

theorem attractor-has-strategy:
  assumes W ⊆ V
  shows ∃ σ. strategy p σ ∧ strategy-attracts p σ (attractor p W) W
⟨proof⟩

end — context ParityGame

end

12 Positional Determinacy of Parity Games

theory PositionalDeterminacy
imports
  Main
  AttractorStrategy
begin

context ParityGame begin

12.1 Induction Step

The proof of positional determinacy is by induction over the size of the finite set ω ‘ V, the set of priorities. The following lemma is the induction step.

For now, we assume there are no deadends in the graph. Later we will get rid of this assumption.

lemma positional-strategy-induction-step:
  assumes v ∈ V
  and no-deadends: ∀ v. v ∈ V → ¬deadend v
  and IH: \( \forall G :: (\langle a, b \rangle \text{ ParityGame-scheme}) \ v. \)
  \[ \text{card} (\omega_G \setminus V_G) < \text{card} (\omega \setminus V); v ∈ V_G; \]
  \[ \text{ParityGame } G; \]
  \[ \forall v. v ∈ V_G → \neg \text{Digraph.deadend } G v \]
  \[ ⇒ ∃ p. v ∈ \text{ParityGame.winning-region } G p \]


12.2 Positional Determinacy without Deadends

\textbf{theorem} positional-strategy-exists-without-deadends:
\begin{itemize}
  \item \textbf{assumes} \( v \in V \land v \in V \implies \neg \text{deadend } v \)
  \item \textbf{shows} \( \exists p. v \in \text{winning-region } p \)
\end{itemize}
\begin{proof}

12.3 Positional Determinacy with Deadends

Prove a stronger version of the previous theorem: Allow deadends.

\textbf{theorem} positional-strategy-exists:
\begin{itemize}
  \item \textbf{assumes} \( v \in V \)
  \item \textbf{shows} \( \exists p. v \in \text{winning-region } p \)
\end{itemize}
\begin{proof}

12.4 The Main Theorem: Positional Determinacy

Prove the main theorem: The winning regions of player \( \text{EVEN} \) and \( \text{ODD} \) are a partition of the set of nodes \( V \).

\textbf{theorem} partition-into-winning-regions:
\begin{itemize}
  \item \textbf{shows} \( V = \text{winning-region } \text{EVEN} \cup \text{winning-region } \text{ODD} \)
  \item \textbf{and} \( \text{winning-region } \text{EVEN} \cap \text{winning-region } \text{ODD} = {} \)
\end{itemize}
\begin{proof}

13 Defining the Attractor with \texttt{inductive_set}

\textbf{theory} AttractorInductive
\begin{verbatim}
  imports Main Attractor
  begin

  context ParityGame begin

  In section 6 we defined \texttt{attractor} manually via \texttt{lfp}. We can also define it with \texttt{inductive_set}. In this section, we do exactly this and prove that the new definition yields the same set as the old definition.

  13.1 \texttt{attractor-inductive}

  The attractor set of a given set of nodes, defined inductively.

end

end

end

\end{verbatim}

34
We show that the inductive definition and the definition via least fixed point are the same.

**Lemma**: `attractor-injective-is-attractor`:

**Assumes**: \( W \subseteq V \)

**Shows**: \( \text{attractor-injective} \ p \ W = \text{attractor} \ p \ W \)

**Proof**

end
References


