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].
4 **Positional Strategies**  
4.1 Definitions ........................................................................ 14  
4.2 Strategy-Conforming Paths ........................................ 14  
4.3 An Arbitrary Strategy .................................................. 15  
4.4 Valid Strategies .......................................................... 15  
4.5 Conforming Strategies ................................................ 16  
4.6 Greedy Conforming Path .......................................... 17  
4.7 Valid Maximal Conforming Paths ......................... 18  
4.8 Valid Maximal Conforming Paths with One Edge .... 19  
4.9 \emph{iset} Induction Schemas for Paths ..................... 20  

5 **Attracting Strategies** ................................................. 20  
5.1 Paths Visiting a Set .................................................... 21  
5.2 Attracting Strategy from a Single Node .................. 21  
5.3 Attracting strategy from a set of nodes ................... 23  

6 **Attractor Sets** ........................................................ 24  
6.1 \emph{directly-attracted} .................................................. 24  
6.2 \emph{attractor-step} ..................................................... 25  
6.3 Basic Properties of an Attractor .......................... 25  
6.4 Attractor Set Extensions ......................................... 25  
6.5 Removing an Attractor ............................................. 25  
6.6 Attractor Set Induction ............................................ 26  

7 **Winning Strategies** .................................................. 26  
7.1 Deadends ................................................................. 27  
7.2 Extension Theorems .................................................. 27  

8 **Well-Ordered Strategy** ............................................. 27  
8.1 Strategies on a Path .................................................... 29  
8.2 Eventually One Strategy ......................................... 30  

9 **Winning Regions** ..................................................... 30  
9.1 Paths in Winning Regions ........................................ 31  
9.2 Irrelevant Updates .................................................... 31  
9.3 Extending Winning Regions .................................... 31  

10 **Uniform Strategies** ................................................ 31  
10.1 A Uniform Attractor Strategy ............................. 32  
10.2 A Uniform Winning Strategy ............................ 32  
10.3 Extending Winning Regions .............................. 32  

11 **Attractor Strategies** .............................................. 32  
11.1 Existence ............................................................... 33  

12 **Positional Determinacy of Parity Games** .......... 33  
12.1 Induction Step ......................................................... 33
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 μ-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 $\text{nat} \Rightarrow \text{'a option}$ with the property that the domain is an initial segment of $\text{nat}$ (and where $\text{'a}$ is the node type). However, it turned out that coinductive lists give simpler proofs.

It is possible to represent a graph as a function $\text{'a} \Rightarrow \text{'a} \Rightarrow \text{bool}$, 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 $\text{UNIV} :: \text{'a set}$ but rather a subset of $\text{UNIV} :: \text{'a set}$, 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 $\text{interpret \ vmc-path G P v0 p \sigma}$ 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.List
begin

2.1 lset

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

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

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

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

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

lemma lfinite-lset: $\text{lfinite} \ xs \implies \neg \text{null} \ xs \implies \text{llast} \ xs \in \text{lset} \ xs$
  (proof)


lemma \textit{lset-subset}: \neg (\textit{lset} \; \textit{xs} \subseteq A) \implies \exists n. \textit{enat} \; n < \textit{llength} \; \textit{xs} \land \textit{lnth} \; \textit{xs} \; n \notin A

\langle \textit{proof} \rangle

2.2 \textit{llength}

lemma \textit{enat-Suc-llt}: 
\textbf{assumes} \textit{enat} \; (\textit{Suc} \; n) < \textit{llength} \; \textit{xs} 
\textbf{shows} \textit{enat} \; n < \textit{llength} \; (\textit{llt} \; \textit{xs})

\langle \textit{proof} \rangle

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

\langle \textit{proof} \rangle

lemma \textit{infinite-small-llength} \; [\textit{intro}]: \neg \textit{finite} \; \textit{xs} \implies \textit{enat} \; n < \textit{llength} \; \textit{xs}

\langle \textit{proof} \rangle

lemma \textit{inval-0-llength}: \neg \textit{inval} \; \textit{xs} \implies \textit{enat} \; 0 < \textit{llength} \; \textit{xs}

\langle \textit{proof} \rangle

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

\langle \textit{proof} \rangle

2.3 \textit{ltake}

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

\langle \textit{proof} \rangle

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

\langle \textit{proof} \rangle

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

\langle \textit{proof} \rangle

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

\langle \textit{proof} \rangle

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

\langle \textit{proof} \rangle

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

\langle \textit{proof} \rangle

2.4 \textit{ldropn}

lemma \textit{llt-ldrop}: \bigwedge x, \; P \; x \implies P \; (\textit{llt} \; x); \; P \; x \implies P \; (\textit{ldropn} \; n \; x)

\langle \textit{proof} \rangle

2.5 \textit{lnite}

lemma \textit{lnite-drop-set}: \textit{lnite} \; \textit{xs} \implies \exists n. \; v \notin \textit{lset} \; (\textit{ldrop} \; n \; \textit{xs})
proof

lemma index-infinite-set:
\[ \neg \text{finite } x ; \text{lnth } x \, \text{m} = y ; \bigwedge i . \text{lnth } x \, i = y \implies (\exists m > i . \text{lnth } x \, m = y) \] \implies y \in \text{set} (\text{ldropn } n \ x)

(proof)

2.6 \text{lmap}

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

(proof)

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

(proof)

2.7 Notation

We introduce the notation \$ to denote \text{lnth}.

notation \text{lnth} (\text{infix } \$ 61)

end

3 Parity Games

theory ParityGame
imports
  Main
  MoreCoinductiveList
begin

3.1 Basic definitions

\text{'a} is the node type. Edges are pairs of nodes.

type-synonym \text{'a Edge} = \text{'a} \times \text{'a}

A path is a possibly infinite list of nodes.

type-synonym \text{'a Path} = \text{'a list}

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 \text{'a Graph} =
  verts :: \text{'a set} (\text{V})
  arcs :: \text{'a Edge set} (\text{E})

abbreviation is-arc :: \text{('a, 'b) Graph-scheme} \Rightarrow \text{'a} \Rightarrow \text{'a} \Rightarrow \text{bool (infixl } \rightarrow 60) \text{ where}
\quad 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 \implies 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-base': v ∈ V ⇒ valid-path (LCons v LNil) \}
  \{ valid-path-cons: [ v ∈ V; w ∈ V; v → w; valid-path Ps; ¬null Ps; lhd Ps = w ] \} ⇒ valid-path (LCons v Ps) 
\{proof\}

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 (ldrop n P) 
\{proof\}

lemma valid-path-in-V: assumes valid-path P shows set 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 I, case-names base step, coinduct pred: valid-path]: 
  \{assumes major: Q P \} 
  \{and base: \∧ v P. Q (LCons v LNil) ⇒ v ∈ V \} 
  \{and step: \∧ 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: 
\[ \text{valid-path } P; \text{enat } i < \text{lenght } P; \text{deadend } (P \# i) \implies \text{enat } (\text{Suc } i) = \text{lenght } P \]
(proof)

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

lemma valid-path-lappend: 
\[ \text{assumes valid-path } P \text{ valid-path } P' \left\{ \text{\neg lnul } P; \text{\neg lnul } P' \right\} \implies \text{last } P \rightarrow \text{hd } P' \]
\[ \text{shows valid-path } \left( \text{lappend } P P' \right) \]
(proof)

A valid path is still valid in a supergame.

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

maximal-path-base: maximal-path LNil
maximal-path-base: \text{deadend } v \implies \text{maximal-path } (LCons v LNil)
maximal-path-cons: \text{\neg lnul } Ps \implies \text{maximal-path } Ps \implies \text{maximal-path } (LCons v Ps)

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

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

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

lemma maximal-path-lappend: 
\[ \text{assumes } \text{\neg lnul } P' \text{ maximal-path } P' \]
\[ \text{shows maximal-path } \left( \text{lappend } P P' \right) \]
(proof)

lemma maximal-ends-on-deadend: 
\[ \text{assumes maximal-path } P \text{ \text{finite } P \neg lnul } P \]
\[ \text{shows deadend } (\text{lend } P) \]
(proof)

lemma maximal-ends-on-deadend': \[ \text{\text{finite } P; deadend } (\text{lend } P) \implies \text{maximal-path } P \]
(proof)

lemma infinite-path-is-maximal: \[ \text{valid-path } P; \neg \text{finite } P \implies \text{maximal-path } P \]
(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 \( p \equiv (\text{if } p = \text{Even} \text{ then Odd else Even}) \)

**notation** other-player \( (\ast \ast) [1000] 1000 \)

**lemma** other-other-player \[ simp \]: \( p \ast \ast \ast = p \) (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 \(V \rightarrow \mathbb{N}\) with finite image.

**record** 'a ParityGame = 'a Graph +
  player0 :: 'a set (V0)
  priority :: 'a \Rightarrow \mathbb{N} (\omega)

**locale** ParityGame = Digraph \( G \) for \( G :: (\text{'a, 'b} \text{ ParityGame}-\text{scheme}) \) +
  assumes valid-player0-set: \( V0 \subseteq V \)
  and priorities-finite: finite \((\omega \cdot V)\)

**begin**

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

**abbreviation** \( VV :: \text{Player} \Rightarrow \text{'a set \( \) where} VV p \equiv (\text{if } p = \text{Even then} V0 \text{ else} V - V0) \)

**lemma** \( VVp-to-V \) [intro]: \( v \in VV p \Rightarrow v \in V \) (proof)

**lemma** \( VV-impl1 \): \( v \in VV p \Rightarrow v \notin VV p** (proof)

**lemma** \( VV-impl2 \): \( v \in VV p** \Rightarrow v \notin VV p \) (proof)

**lemma** \( VV-equivalence \) [iff]: \( v \in V \Rightarrow v \notin VV p \iff v \in VV p** \) (proof)

**lemma** \( VV-cases \) [consumes 1]: \( [v \in V : v \in VV p \Rightarrow P ; v \in VV p** \Rightarrow P ] \Rightarrow P \) (proof)

3.6 Sets of Deadends

**definition** deadends \( p \equiv \{ v \in VV p, \text{ deadend} \} \)

**lemma** deadends-in-V: deadends \( p \subseteq V \) (proof)

3.7 Subgames

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

**definition** subgame where

\( \text{subgame} \ V':: \text{G} \)

\( \text{verbs} := V \cap V' \)

\( \text{arcs} := E \cap (V' \times V') \)

\( \text{player0} := V0 \cap V' \)

**lemma** \( \text{subgame}-V \) [simp]: \( V_{\text{subgame}} V' \subseteq V \)

**and** \( \text{subgame}-E \) [simp]: \( E_{\text{subgame}} V' \subseteq E \)

**and** \( \text{subgame}-\omega : \text{\( \omega_{\text{subgame}} V' = \omega \)} \) (proof)

**lemma**

**assumes** \( V' \subseteq V \)

**shows** \( \text{subgame}-V' \) [simp]: \( V_{\text{subgame}} V' = V' \)
and $\text{subgame-}E'$ [simp]: $E_{\text{subgame } V'} = E \cap (V_{\text{subgame } V'} \times V_{\text{subgame } V'})$

\text{lemma subgame-}VV$ [simp]: $\text{ParityGame}.VV (\text{subgame } V') p = V' \cap VV p \langle \text{proof} \rangle$

\text{corollary subgame-}VV-subset [simp]: $\text{ParityGame}.VV (\text{subgame } V') p \subseteq VV p \langle \text{proof} \rangle$

\text{lemma subgame-finite [simp]: finite } (\omega_{\text{subgame } V'} \cdot \cdot V_{\text{subgame } V'}) \langle \text{proof} \rangle$

\text{lemma subgame-}ω-subset [simp]: $\omega_{\text{subgame } V'} \cdot \cdot V_{\text{subgame } V'} \subseteq \omega \cdot V$

\text{lemma subgame-Digraph}: $\text{Digraph } (\text{subgame } V')$

\text{lemma subgame-}ParityGame:

\text{shows } ParityGame (\text{subgame } V')\langle \text{proof} \rangle$

\text{lemma subgame-valid-path}:

\text{assumes } P: valid-path P lset P \subseteq V'

\text{shows } Digraph.valid-path (\text{subgame } V') P \langle \text{proof} \rangle$

\text{lemma subgame-maximal-path}:

\text{assumes } V': V' \subseteq V and P: maximal-path P lset P \subseteq V'

\text{shows } Digraph.maximal-path (\text{subgame } V') P \langle \text{proof} \rangle$

\text{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.

\text{definition path-inf-priorities :: 'a Path ⇒ nat set where}

\hspace{1cm} path-inf-priorities P ≡ \{k. \forall n. k \in lset (ldrop 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.

\text{lemma path-inf-priorities-is-nonempty}:

\text{assumes } P: valid-path P lfinite P

\text{shows } \exists k. k \in path-inf-priorities P \langle \text{proof} \rangle$

\text{lemma path-inf-priorities-at-least-min-prio}:

\text{assumes } P: valid-path P and a: a \in path-inf-priorities P

\text{shows } Min (\omega \cdot V) \leq a \langle \text{proof} \rangle$

\text{lemma path-inf-priorities-LCons}:

\text{path-inf-priorities } P = path-inf-priorities (LCons v P) [\text{is } ?A = ?B] \langle \text{proof} \rangle
**Corollary** path-inf-priorities-lll: path-inf-priorities $P = \text{path-inf-priorities (}$\ltl P)$

### 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 $p \equiv (\text{if } p = \text{Even then even else odd})$

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

- winning-path $p$ $P \equiv ($\neg$\text{finite } P \land (\exists a \in \text{path-inf-priorities } P.$
- $(\forall b \in \text{path-inf-priorities } P. a \leq b) \land \text{winning-priority } p a)$
- $\lor (\neg\text{null } P \land \text{finite } P \land \text{last } P \in \text{VV } p^{**})$
- $\lor (\text{null } P \land p = \text{Even})$

Every path has a unique winner.

**Lemma** paths-are-winning-for-one-player:

- **Assumes** valid-path $P$
- **Shows** winning-path $p$ $P \iff \neg$winning-path $p^{**}$ $P$

**Proof**

**Lemma** winning-path-lll:

- **Assumes** $P$: winning-path $p$ $P \neg\text{null } P \neg\text{null } (\ltl P)$
- **Shows** winning-path $p$ ($\ltl P$)

**Proof**

**Corollary** winning-path-drop:

- **Assumes** winning-path $p$ $P \text{ enat } n < \text{length } P$
- **Shows** winning-path $p$ ($\text{dropn } n F$)

**Proof**

**Corollary** winning-path-drop-add:

- **Assumes** valid-path $P$ winning-path $p$ ($\text{dropn } n F$) $\text{enat } n < \text{length } P$
- **Shows** winning-path $p$ $P$

**Proof**

**Lemma** winning-path-LCons:

- **Assumes** $P$: winning-path $p$ $P \neg\text{null } P$
- **Shows** winning-path $p$ ($\text{LCons } v P$)

**Proof**

**Lemma** winning-path-supergame:

- **Assumes** winning-path $p$ $P$
- **And** $G': \text{ParityGame } G' \text{ VV } p^{**} \subseteq \text{ParityGame.VV } G' \text{ p** } \omega = \omega_{G'}$
- **Shows** $\text{ParityGame.winning-path } G' \text{ p } P$

**Proof**
end — locale ParityGame

3.10 Valid Maximal Paths

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

locale vm-path = ParityGame +
  fixes P v0
  assumes P-not-null [simp]: ¬null 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 (lL P) (proof)

lemma P-len [simp]: enat 0 < llength P (proof)
lemma P-0 [simp]: P $ 0 = v0 (proof)
lemma P-Inth-Suc: P $ Suc n = lL P $ n (proof)
lemma P-no-deadends: enat (Suc n) < llength P ⇒ ¬deadend (P $ n) (proof)
lemma P-no-deadend-v0: ¬null (lL P) ⇒ ¬deadend v0 (proof)
lemma P-no-deadend-v0-length: 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-tll-deadend-v0: lnull (lL P) ⇒ deadend v0 (proof)
lemma P-lnull-tll-LCons: lnull (lL P) ⇒ P = LCons v0 LNil (proof)
lemma P-deadend-v0-LCons: deadend v0 ⇒ P = LCons v0 LNil (proof)

lemma P-ltake-valid [simp]: valid-path (ltake n P) (proof)
lemma P-ltake-maximal [simp]: maximal-path (ltake n P) (proof)

lemma P-drop-valid [simp]: valid-path (ldrop n P) (proof)
lemma P-drop-maximal [simp]: maximal-path (ldrop 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)
If a path visits a deadend, it is winning for the other player.

lemma visits-deadend: assumes lset P ∩ de adends p ≠ {} shows 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 ⇒ 'a

A valid strategy for player p is a function assigning a successor to each node in VV p.

definition (in ParityGame) strategy :: Player ⇒ 'a Strategy ⇒ bool where
strategy p σ ≡ ∀ v ∈ VV p. ¬deadend v → v→σ v

lemma (in ParityGame) strategyI [intro]:
(∀ v. v ∈ VV p; ¬deadend v ) → v→σ v) ⇒ strategy p σ
⟨proof⟩

4.2 Strategy-Conforming Paths

If path-conforms-with-strategy p P σ holds, then we call P a σ-path. This means that P follows σ on all nodes of player p except maybe the last node on the path.

coinductive (in ParityGame) path-conforms-with-strategy :: Player ⇒ 'a Path ⇒ 'a Strategy ⇒ bool where
  path-conforms-LNil: path-conforms-with-strategy p LNil σ
  | path-conforms-LCons-LNil: path-conforms-with-strategy p (LCons v LNil) σ

path-conforms-Vv: \[ v \in VV_p; w = \sigma v; \text{path-conforms-with-strategy } p \ (LCons \ w \ Ps) \ \sigma \] 
\[ \implies \ \text{path-conforms-with-strategy } p \ (LCons \ v \ (LCons \ w \ Ps)) \ \sigma \]
path-conforms-Vvstar: \[ v \notin VV_p; \text{path-conforms-with-strategy } p \ Ps \ \sigma \]
\[ \implies \ \text{path-conforms-with-strategy } p \ (LCons \ v \ Ps) \ \sigma \]

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 \( vmc-path = \text{vm-path} + \)
\[ \text{fixes } p \ \sigma \ \text{assumes } \text{F-conforms} \ [\text{simp}]: \text{path-conforms-with-strategy } p \ P \ \sigma \]

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

locale \( \text{vmc2-path = comp?}: \text{vmc-path } G \ P \ \nu0 \ P \ \sigma \ast + \text{vmc-path } G \ P \ \nu0 \ P \ \sigma \)
for \( G \ P \ \nu0 \ P \ \sigma \ \sigma' \)

4.3 An Arbitrary Strategy

class \( \text{ParityGame} \)

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

definition \( \sigma\text{-arbitrary } \equiv \lambda v. \ \text{SOME } w. v \to w \)

lemma \( \text{valid-arbitrary-strategy} \ [\text{simp}]: \text{strategy } p \ \sigma\text{-arbitrary} \ (\text{proof}) \)

4.4 Valid Strategies

lemma \( \text{valid-strategy-updates}: \ [\text{strategy } p \ \sigma; \ \nu0 \to \nu0] \implies \text{strategy } p \ (\sigma(\nu0 := \nu0)) \)
(\text{proof})

lemma \( \text{valid-strategy-updates-set}: \)
\[ \text{assumes } \text{strategy } p \ \sigma \ \land \ \forall v. \ [v \in A; v \in VV_p; \ \neg \text{deadend } v] \implies v \to \sigma' \ v \]
shows \( \text{strategy } p \ (\text{override-on } \sigma \ \sigma' \ A) \)
(\text{proof})

lemma \( \text{valid-strategy-updates-set-strong}: \)
\[ \text{assumes } \text{strategy } p \ \sigma \ \text{strategy } p \ \sigma' \]
shows \( \text{strategy } p \ (\text{override-on } \sigma \ \sigma' \ A) \)
(\text{proof})

lemma \( \text{subgame-strategy-stays-in-subgame}: \)
\[ \text{assumes } \sigma: \text{ParityGame} . \text{strategy} \ (\text{subgame } V') \ p \ \sigma \]
and \( v \in \text{ParityGame} . VV \ (\text{subgame } V') \ p \ \neg \text{Digraph.deadend } (\text{subgame } V') \ v \)
shows \( \sigma \ v \in V' \)
(\text{proof})

lemma \( \text{valid-strategy-supergame}: \)
\[ \text{assumes } \sigma: \text{strategy } p \ \sigma \]
and \( \sigma': \text{ParityGame} . \text{strategy} \ (\text{subgame } V') \ p \ \sigma' \]
and \( G'\text{-no-deadends}: \ \forall v. v \in V' \implies \neg \text{Digraph.deadend } (\text{subgame } V') \ v \)
shows strategy p (override-on σ σ' V') (is strategy p ?σ)
(proof)

lemma valid-strategy-in-V: [ strategy p σ; v ∈ VV p; ¬deadend v ] ⇒ σ v ∈ V
(proof)

lemma valid-strategy-only-in-V: [ strategy p σ; \(\bigwedge_v v ∈ V ⇒ σ v = σ' v\) ] ⇒ strategy p σ'
(proof)

4.5 Conforming Strategies

lemma path-conforms-with-strategy-ltl [intro]:
path-conforms-with-strategy p P σ ⇒ path-conforms-with-strategy p (ltl P) σ
(proof)

lemma path-conforms-with-strategy-drop:
path-conforms-with-strategy p P σ ⇒ path-conforms-with-strategy p (ldrop n P) σ
(proof)

lemma path-conforms-with-strategy-prefix:
path-conforms-with-strategy p P σ ⇒ lprefix P' P ⇒ path-conforms-with-strategy p P' σ
(proof)

lemma path-conforms-with-strategy-irrelevant:
assumes path-conforms-with-strategy p P σ v ∉ lset P
shows path-conforms-with-strategy p P (σ(v := w))
(proof)

lemma path-conforms-with-strategy-irrelevant-deadend:
assumes path-conforms-with-strategy p P σ deadend v ∨ v ∉ VV p valid-path P
shows path-conforms-with-strategy p P (σ(v := w))
(proof)

lemma path-conforms-with-strategy-irrelevant-updates:
assumes path-conforms-with-strategy p P σ \(\bigwedge_v v ∈ lset P ⇒ σ v = σ' v\)
shows path-conforms-with-strategy p P σ'
(proof)

lemma path-conforms-with-strategy-irrelevant':
assumes path-conforms-with-strategy p P (σ(v := w)) v ∉ lset P
shows path-conforms-with-strategy p P σ
(proof)

lemma path-conforms-with-strategy-irrelevant-deadend':
assumes path-conforms-with-strategy p P (σ(v := w)) deadend v ∨ v ∉ VV p valid-path P
shows path-conforms-with-strategy p P σ
(proof)

lemma path-conforms-with-strategy-start:
path-conforms-with-strategy p (LCons v (LCons w P)) σ ⇒ v ∈ VV p ⇒ σ v = w
(proof)
lemma path-conforms-with-strategy-append:
  assumes
  P: lfinite P ¬null P path-conforms-with-strategy p P σ
  and P': ¬null P' path-conforms-with-strategy p P' σ
  and conforms: |last P ∈ VV p ||→ σ (|last P) = lid P'
  shows path-conforms-with-strategy p (append P P') σ
 ⟨proof⟩

lemma path-conforms-with-strategy-VVpstar:
  assumes lset P ⊆ VV p**
  shows path-conforms-with-strategy p P σ
 ⟨proof⟩

lemma subgame-path-conforms-with-strategy:
  assumes V': V' ⊆ V and P: path-conforms-with-strategy p P σ lset P ⊆ V'
  shows ParityGame.path-conforms-with-strategy (subgame V') p P σ
 ⟨proof⟩

lemma (in vmc-path) subgame-path-vmc-path:
  assumes V': V' ⊆ V and P: lset P ⊆ V'
  shows vmc-path (subgame V') P v0 p σ
 ⟨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.

corollary greedy-path-deadend-v:
  greedy-conforming-path p σ σ' v0 = LCons v LNil \rightarrow\ \mathrm{deadend} v
 ⟨proof⟩

corollary greedy-path-deadend-v':
  greedy-conforming-path p σ σ' v0 = LCons v LNil \rightarrow\ \mathrm{deadend} v
 ⟨proof⟩
\begin{proof} \textbf{lemma} greedy-path-ll: 
\textbf{assumes} greedy-conforming-path p \sigma \sigma' v0 = LCons v P 
\textbf{shows} P = LNil \lor P = greedy-conforming-path p \sigma \sigma' P 
\end{proof}

\begin{proof} \textbf{lemma} greedy-path-ll-ex: 
\textbf{assumes} greedy-conforming-path p \sigma \sigma' v0 = LCons v P 
\textbf{shows} P = LNil \lor (\exists v. P = \text{greedy-conforming-path} p \sigma \sigma' v) 
\end{proof}

\begin{proof} \textbf{lemma} greedy-path-ll-\text{VVp}: 
\textbf{assumes} greedy-conforming-path p \sigma \sigma' v0 = LCons v P v0 \in \text{VVp} 
\textbf{shows} \sigma v0 = lhd P 
\end{proof}

\begin{proof} \textbf{lemma} greedy-path-ll-\text{VVp\ast}: 
\textbf{assumes} greedy-conforming-path p \sigma \sigma' v0 = LCons v P v0 \in \text{VVp}\ast 
\textbf{shows} \sigma' v0 = lhd P 
\end{proof}

\begin{proof} \textbf{lemma} greedy-conforming-path-properties: 
\textbf{assumes} v0 \in \text{V strategy p strategy p**} \sigma' 
\textbf{shows} 
\begin{align*}
\text{greedy-path-not-null: } & \neg \text{null} (\text{greedy-conforming-path} p \sigma \sigma' v0) \\
\text{and greedy-path-v0: } & \text{greedy-conforming-path} p \sigma \sigma' v0 \& v0 = v0 \\
\text{and greedy-path-valid: } & \text{valid-path} (\text{greedy-conforming-path} p \sigma \sigma' v0) \\
\text{and greedy-path-maximal: } & \text{maximal-path} (\text{greedy-conforming-path} p \sigma \sigma' v0) \\
\text{and greedy-path-conforms: } & \text{path-conforms-with-strategy} p (\text{greedy-conforming-path} p \sigma \sigma' v0) \sigma' \\
\text{and greedy-path-conforms': } & \text{path-conforms-with-strategy} p** (\text{greedy-conforming-path} p \sigma \sigma' v0) \sigma' \\
\end{align*}
\end{proof}

\begin{corollary} strategy-conforming-path-exists: 
\textbf{assumes} v0 \in V \text{ strategy p strategy p**} \sigma' 
\textbf{obtains} P \text{ where } \text{vmc-path G P v0 p } \sigma' 
\end{corollary}

\begin{corollary} strategy-conforming-path-exists-single: 
\textbf{assumes} v0 \in V \text{ strategy p } \sigma 
\textbf{obtains} P \text{ where } \text{vmc-path G P v0 p } \sigma 
\end{corollary}

4.7 Valid Maximal Conforming Paths

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

\begin{proof} \textbf{context} vmc-path \textbf{begin} 
\textbf{lemma} \textit{Ptl-conforms} \textbf{[simp]}: \textit{path-conforms-with-strategy} p (\textit{ltl} P) \sigma 
\end{proof}

\begin{proof} \textbf{lemma} \textit{Pdrop-conforms} \textbf{[simp]}: \textit{path-conforms-with-strategy} p (\textit{ldropn} n P) \sigma 
\end{proof}

\begin{proof} \textbf{lemma} \textit{prefix-conforms} \textbf{[simp]}: \textit{path-conforms-with-strategy} p (\textit{ltake} n P) \sigma 
\end{proof}

\begin{proof} \textbf{lemma} \textit{extension-conforms} \textbf{[simp]}: 
(\exists v' \in \text{VVp} \implies \sigma v' = v0) \implies \textit{path-conforms-with-strategy} p (\textit{LCons} v' P) \sigma 
\end{proof}
lemma extension-valid-maximal-conforming:
assumes \( v' \rightarrow v_0 \) \( v' \in VV P \implies \sigma v' = v_0 \)
shows \( \text{vmc-path } G \ (LCons v' P) \ v' p \ \sigma \)
⟨proof⟩

lemma \( \text{vmc-path-ldropn} \):
assumes \( \text{enat } n < \text{length } P \)
shows \( \text{vmc-path } G \ (\text{ldropn } n \ P) \ (P \ \& n) \ p \ \sigma \)
⟨proof⟩

lemma conforms-to-another-strategy:
path-conforms-with-strategy \( P \ \sigma' \implies \text{vmc-path } G \ P \ v_0 \ p \ \sigma' \)
⟨proof⟩

end

lemma (in ParityGame) valid-maximal-conforming-path-0:
assumes \( \neg \text{null } P \) valid-path \( P \) maximal-path \( P \) path-conforms-with-strategy \( P \ \sigma \)
shows \( \text{vmc-path } G \ P \ (P \ \& 0) \ p \ \sigma \)
⟨proof⟩

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 vmc-path-no-deadend = vmc-path +
assumes \( \text{v0-no-deadend } \) simp \( \neg \text{deadend } v_0 \)
begin
definition \( w_0 \equiv \text{lhd } (\text{ltl } P) \)

lemma \( \text{Plt-not-null } \) simp \( \neg \text{null } (\text{ltl } P) \)
⟨proof⟩

lemma \( \text{Plt-LCons} : \text{ltl } P = \text{LCons } w_0 \ (\text{ltl } (\text{ltl } P)) \) ⟨proof⟩

lemma \( \text{P-LCons} : P = \text{LCons } v_0 \ (\text{LCons } w_0 \ (\text{ltl } (\text{ltl } P))) \) ⟨proof⟩

lemma \( v_0-edge-w_0 \) simp \( v_0 \rightarrow w_0 \) ⟨proof⟩

lemma \( \text{Plt-0} : \text{ltl } P \ \& 0 = \text{lhd } (\text{ltl } P) \) ⟨proof⟩

lemma \( \text{P-Suc-0} : P \ \& \text{Suc } 0 = w_0 \) ⟨proof⟩

lemma \( \text{Plt-edge } \) simp \( v_0 \rightarrow \text{lhd } (\text{ltl } P) \) ⟨proof⟩

lemma \( v_0-conforms : v_0 \in VV p \implies \sigma v_0 = w_0 \)
⟨proof⟩

lemma \( w_0-V \) simp \( w_0 \in V \) ⟨proof⟩

lemma \( w_0-lset-P \) simp \( w_0 \in \text{lset } P \) ⟨proof⟩

lemma \( \text{vmc-path-ltl } \) simp \( \text{vmc-path } G \ (\text{ltl } P) \ w_0 p \ \sigma \) ⟨proof⟩
end
context vmc-path begin

lemma vmc-path-hull-llt-no-deadend:
\( \neg \text{hull} (\text{llt} \, P) \implies \text{vmc-path-no-deadend} \, G \, P \, v0 \, p \, \sigma \)
(proof)

lemma vmc-path-conforms:
assumes enat (Suc \( n \)) < length \( P \) \( P \, \$ \, n \in \text{VV} \, p \)
shows \( \sigma (P \, \$ \, n) = P \, \$ \, \text{Suc} \, n \)
(proof)

4.9 \( \text{let} \) Induction Schemas for \( \text{Paths} \)

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: \( v0 \in \text{S} \)
and step-assumption: \( \forall v0. [ \text{vmc-path-no-deadend} \, G \, P \, v0 \, p \, \sigma ; \, v0 \in \text{S} ; \, Q \, P ] \implies Q (\text{llt} \, P) \wedge (\text{vmc-path-no-deadend}\, w0 \, P) \in \text{S} \)
shows lset \( P \subseteq \text{S} \)
(proof)

\[ ?Q \, P ; \, v0 \in ?S ; \, \forall v0. [ \text{vmc-path-no-deadend} \, G \, v0 \, p \, \sigma ; \, v0 \in ?S ; \, ?Q \, P ] \implies \neg \text{Q (llt} \, P) \wedge \text{vmc-path-no-deadend}, \text{w0 P} \in \text{S} ] \implies \text{lset} \, P \subseteq ?S \text{ without the Q predicate.} \)

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

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

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

end

end

5 Attracting Strategies

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

**5.1 Paths Visiting a Set**

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

**definition** $\text{visits-via} \; P \; A \; W \equiv \exists \; n. \; \text{enat} \; n < \text{length} \; P \wedge P \; \$ \; n \in W \wedge \text{let} \; \text{ltake} \; (\text{enat} \; n) \; P \subseteq A$

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

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

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

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

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

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

**5.2 Attracting Strategy from a Single Node**

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

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

**lemma** ($\text{in vmc-path}$) $\text{strategy-attracts-via-E}$:
- **assumes** $\text{strategy-attracts-via} \; p \; \sigma \; v0 \; A \; W$
- **shows** $\text{visits-via} \; P \; A \; W$

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

21
\begin{proof}

**Lemma (in vmc-path) strategy-attracts-via-lset:**
- **Assumes** strategy-attracts-via \( p \sigma v0 A W \)
- **Shows** \( \text{set } P \cap W \neq \{\} \)

\end{proof}

\begin{proof}

**Lemma strategy-attracts-via-v0:**
- **Assumes** \( \sigma; \text{strategy } p \sigma \text{ strategy-attracts-via } p \sigma v0 A W \)
- and \( v0: v0 \in V \)
- **Shows** \( v0 \in A \cup W \)

\end{proof}

**Corollary strategy-attracts-not-outside:**
\[
\begin{align*}
[ v0 \in V - A - W; \text{strategy } p \sigma ] & \implies \neg\text{strategy-attracts-via } p \sigma v0 A W \\
\end{align*}
\]

\end{proof}

\begin{proof}

**Lemma strategy-attracts-via1 [intro]:**
- **Assumes** \( \bigwedge P. \text{vmc-path } G P v0 p \sigma \implies \text{visits-via } P A W \)
- **Shows** strategy-attracts-via \( p \sigma v0 A W \)

\end{proof}

\begin{proof}

**Lemma strategy-attracts-via-no-deadends:**
- **Assumes** \( v \in V \) \( v \in A - W \) \( \text{strategy-attracts-via } p \sigma v A W \)
- **Shows** \( \neg\text{deadend } v \)

\end{proof}

\begin{proof}

**Lemma attractor-strategy-on-extends:**
\[
\begin{align*}
[ \text{strategy-attracts-via } p \sigma v0 A W; A \subseteq A' ] & \implies \text{strategy-attracts-via } p \sigma v0 A' W \\
\end{align*}
\]

\end{proof}

\begin{proof}

**Lemma strategy-attracts-via-trivial:** \( v0 \in W \implies \text{strategy-attracts-via } p \sigma v0 A W \)

\end{proof}

\begin{proof}

**Lemma strategy-attracts-via-successor:**
- **Assumes** \( \sigma; \text{strategy } p \sigma \text{ strategy-attracts-via } p \sigma v0 A W \)
- and \( v0: v0 \in A - W \)
- and \( w0: v0 \rightarrow w0 v0 \in VV p \implies \sigma v0 = w0 \)
- **Shows** strategy-attracts-via \( p \sigma w0 A W \)

\end{proof}

\begin{proof}

**Lemma strategy-attracts-VVp:**
- **Assumes** \( v \in A - W \) \( v0 \in VV \) \( \neg\text{deadend } v0 \)
- **Shows** \( \sigma v0 \in A \cup W \)

\end{proof}

\begin{proof}

**Lemma strategy-attracts-VVpstar:**
- **Assumes** \( v0 \in A - W \) \( v0 \notin VV \) \( w0 \in V - A - W \)
- **Shows** \( \neg v0 \rightarrow w0 \)

\end{proof}
5.3 Attracting strategy from a set of nodes

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

**definition** strategy-attracts :: Player \( \Rightarrow \) 'a Strategy \( \Rightarrow \) 'a set \( \Rightarrow \) bool where

\[
\text{strategy-attracts} \ p \ \sigma \ A \ W \equiv \forall v \in A. \ \text{strategy-attracts-via} \ p \ \sigma \ v \in A \ W
\]

**lemma** (in vmc-path) strategy-attractsE:

\[
\text{assumes} \ \text{strategy-attracts} \ p \ \sigma \ A \ W \ v \in A
\]

\[
\text{shows} \ \text{visits-via} \ P \ A \ W
\]

(proof)

**lemma** strategy-attractsI [intro]:

\[
\text{assumes} \ \forall v. [ v \in A; \ \text{vmc-path} \ G \ v \ p \ \sigma ] \implies \text{visits-via} \ P \ A \ W
\]

\[
\text{shows} \ \text{strategy-attracts} \ p \ \sigma \ A \ W
\]

(proof)

**lemma** (in vmc-path) strategy-attracts-set:

\[
\text{assumes} \ \text{strategy-attracts} \ p \ \sigma \ A \ W \ v \in A
\]

\[
\text{shows} \ \text{set} \ P \ \cap \ W \neq \{\}
\]

(proof)

**lemma** strategy-attracts-empty [simp]: strategy-attracts \( p \ \sigma \ \{\} \ W \)

(proof)

**lemma** strategy-attracts-invalid-path:

\[
\text{assumes} \ P : P = \text{LCons} \ v (\text{LCons} \ w \ P') \ v \in A - W \ w \notin A \cup W
\]

\[
\text{shows} \ \neg \text{visits-via} \ P \ A \ W \ (\text{is} \ \neg \ A)
\]

(proof)

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

**lemma** strategy-attracts-does-not-leave:

\[
\text{assumes} \ \sigma : \text{strategy-attracts} \ p \ \sigma \ A \ W \text{ strategy} \ p \ \sigma
\]

\[
\text{and} \ v : v \rightarrow w \ v \in A - W \ w \notin A \cup W
\]

\[
\text{shows} \ v \in V V p \land \sigma \ v \neq w
\]

(proof)

Given an attracting strategy \( \sigma \), we can turn every strategy \( \sigma' \) into an attracting strategy by overriding \( \sigma' \) 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-ignore:

\[
\text{assumes} \ \text{strategy-attracts} \ p \ \sigma \ A \ W \text{ strategy} \ p \ \sigma
\]

\[
\text{shows} \ \text{strategy-attracts} \ p \ \text{(ignore-on} \ \sigma' \ \sigma \ (A - W)) \ A \ W
\]

(proof)

**lemma** strategy-attracts-trivial [simp]: strategy-attracts \( p \ \sigma \ W \ W \)

(proof)

If a \( \sigma \)-conforming path \( P \) hits an attractor \( A \), it will visit \( W \).

**lemma** (in vmc-path) attracted-path:

\[
\text{assumes} \ W \subseteq V
\]
and $\sigma$: strategy-attracts $p \sigma A W$
and $P$-hits-$A$: set $P \cap A \neq \{\}$
shows set $P \cap W \neq \{\}$
(proof)

lemma attracted-strategy-step:
assumes $\sigma$: strategy $p \sigma$ strategy-attracts $p \sigma A W$
and $v0$: $\neg$deadend $v0 v0 \in A - W v0 \in VV p$
shows $\sigma v0 \in A \cup W$
(proof)

lemma (in vmc-path-no-deadend) attracted-path-step:
assumes $\sigma$: strategy-attracts $p \sigma A W$
and $v0$: $v0 \in A - W$
shows $w0 \in A \cup W$
(proof)

end — context ParityGame

end

6 Attractor Sets

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 $v \wedge$
  $(v \in VV p \longrightarrow (\exists w. v \rightarrow w \wedge w \in S))$
  $\wedge (v \in VV p^{**} \longrightarrow (\forall w. v \rightarrow w \longrightarrow w \in S))\}$

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 $p W = lfp$ (attractor-step $p W$)

6.1 directly-attracted

Show a few basic properties of directly-attracted.

lemma directly-attracted-disjoint [simp]: directly-attracted $p W \cap W = \{\}$
and directly-attracted-empty [simp]: directly-attracted $p \{\} = \{\}$
6.2 attractor-step

Lemma attractor-step-empty: attractor-step $\{\} \{\} = \{\}$

and attractor-step-bounded-by-V: $[ W \subseteq V; S \subseteq V ] \implies$ attractor-step $p W S \subseteq V$

(proof)

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

Lemma attractor-step-mono: mono (attractor-step $p W$)

(proof)

6.3 Basic Properties of an Attractor

Lemma attractor-unfolding: attractor $p W =$ attractor-step $p W$ (attractor $p W$)

(proof)

Lemma attractor-lowerbound: attractor-step $p W S \subseteq S \implies$ attractor $p W \subseteq S$

(proof)

Lemma attractor-set-non-empty: $W \neq \{\} \implies$ attractor $p W \neq \{\}$

and attractor-set-base: $W \subseteq$ attractor $p W$

(proof)

Lemma attractor-in-V: $W \subseteq V \implies$ attractor $p W \subseteq V$

(proof)

6.4 Attractor Set Extensions

Lemma attractor-set-VP:

- assumes $v \in V V p v \rightarrow w w \in$ attractor $p W$
- shows $v \in$ attractor $p W$

(proof)

Lemma attractor-set-VPstar:

- assumes $\neg$deadend $v \land w, v \rightarrow w \implies w \in$ attractor $p W$
- shows $v \in$ attractor $p W$

(proof)

6.5 Removing an Attractor

Lemma removing-attractor-induces-no-deadends:

- assumes $v \in S -$ attractor $p W v \rightarrow w w \in S \land w . [ v \in V V p * * ; v \rightarrow w ] \implies w \in S$
- shows $\exists w \in S -$ attractor $p W . v \rightarrow w$

(proof)

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

Lemma subgame-without-deadends:

- assumes $V ' \text{def: } V ' = V -$ attractor $p$ (deadends $p * * * -$ attractor $p * *$ (deadends $p * * * *$)
  (is $V ' = V - ?A - ?B$)
- and $v: v \in V ' _{\text{subgame } V '}$
6.6 Attractor Set Induction

**lemma** mono-restriction-is-mono: \( \text{mono } f \implies \text{mono } (\lambda S. f (S \cap V)) \)

(proof)

Here we prove a powerful induction schema for attractor. Being able to prove this is the only reason why we do not use 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: \( \forall S, S \subseteq V \implies P \quad S \implies P \quad (\text{attractor-step } p \quad W \quad S) \)

and union: \( \forall M, \forall S \in M. \quad S \subseteq V \quad \land \quad P \quad S \implies P \quad (\bigcup M) \)

shows \( P \quad (\text{attractor } p \quad W) \)

(proof)

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 \( p \) \( \sigma' \) winning-strategy \( p \) \( \sigma' \) \( v \)
and \( v \): \( v \in V \)
shows False 
(proof)

7.1 Deadends

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

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

7.2 Extension Theorems

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

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

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

lemma strategy-extends-backwards-VVp:
assumes \( v0 \): \( v0 \in V \) \( \sigma \)
and \( \sigma \): strategy \( p \) \( \sigma \) \( \wedge \ w \). \( \sigma \ v0 \) \( \rightarrow \) \( w \)
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.

```lean
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 :: ('a 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 {σ. ∃ v ∈ S. σ ∈ good v} r
  — Every node has a good strategy.
  and good-ex: ∀ v. v ∈ S ⇒ ∃ σ. σ ∈ good v
  — good strategies are well-formed strategies.
  and good-strategies: ∀ v σ. σ ∈ good v ⇒ strategy p σ
  — 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 ≡ {σ. ∃ 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 (λv. 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)
```
choose always chooses a minimal good strategy on $S$.

**Lemma** choose-works:
assumes $v \in S$
shows minimal-good-strategy $v$ (choose $v$)
(proof)

**Corollary**
assumes $v \in S$
shows choose-good: choose $v \in$ good $v$
and choose-minimal: $\forall \sigma', (\sigma', \text{choose } v) \in r - \text{Id} \implies \sigma' \notin \text{good } v$
and choose-strategy: strategy $p$ (choose $v$)
(proof)

**Corollary** choose-in-Strategies: $v \in S \implies$ choose $v \in$ Strategies (proof)

**Lemma** well-ordered-strategy-valid: strategy $p$ well-ordered-strategy
(proof)

### 8.1 Strategies on a Path

Maps a path to its strategies.

**Definition** path-strategies $\equiv$ lmap choose

**Lemma** path-strategies-in-Strategies:
assumes let $P \subseteq S$
shows let (path-strategies $P$) $\subseteq$ Strategies
(proof)

**Lemma** path-strategies-good:
assumes let $P \subseteq S$ enat $n < \text{length } P$
shows path-strategies $P$ $\& n \in$ good $(P \& n)$
(proof)

**Lemma** path-strategies-strategy:
assumes let $P \subseteq S$ enat $n < \text{length } P$
shows strategy $p$ (path-strategies $P$ $\& n$)
(proof)

**Lemma** path-strategies-monotone-Suc:
assumes $P$; let $P \subseteq S$ valid-path $P$ path-conforms-with-strategy $p$ $P$ well-ordered-strategy enat $(\text{Suc } n) < \text{length } P$
shows (path-strategies $P$ $\& \text{Suc } n$, path-strategies $P$ $\& n$) $\in$ $r$
(proof)
\textbf{lemma} path-strategies-monotone:
\begin{itemize}
  \item \textbf{assumes} \( P \subseteq S \) valid-path \( P \) path-conforms-with-strategy \( p \) \( P \) well-ordered-strategy 
  \item \textbf{shows} (path-strategies \( P \) $ m $, path-strategies \( P \) $ n $) $ \in $ \( r \)
\end{itemize}

\begin{proof}

\end{proof}

\textbf{lemma} path-strategies-eventually-constant:
\begin{itemize}
  \item \textbf{assumes} \( \neg \) finite \( P \) \( P \subseteq S \) valid-path \( P \) path-conforms-with-strategy \( p \) \( P \) well-ordered-strategy 
  \item \textbf{shows} \( \exists \) \( n \). \( \forall m \geq n \). path-strategies \( P \) $ n $ = path-strategies \( P \) $ m $
\end{itemize}

\begin{proof}

\end{proof}

\subsection{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.

\textbf{lemma} path-eventually-conforms-to-\( \sigma \)-map-\( n \):
\begin{itemize}
  \item \textbf{assumes} \( P \subseteq S \) valid-path \( P \) path-conforms-with-strategy \( p \) \( P \) well-ordered-strategy 
  \item \textbf{shows} \( \exists \) \( n \). path-conforms-with-strategy \( p \) (\( \text{drop} \) \( n \) \( P \) \( \text{path-strategies} \) \( P \) $ n \) \( \text{path-strategies} \) \( P \) $ m $)
\end{itemize}

\begin{proof}

\end{proof}

\section{9 Winning Regions}

\textbf{theory} WinningRegion
\textbf{imports}
  Main
  WinningStrategy
\textbf{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.

\textbf{context} ParityGame \textbf{begin}

\textbf{definition} winning-region \( p \equiv \{ v \in V. \exists \sigma. \text{strategy} \ p \ \sigma \ \land \ \text{winning-strategy} \ p \ \sigma \ \langle v \rangle \} \)

\textbf{lemma} winning-regionI [intro]:
\begin{itemize}
  \item \textbf{assumes} \( v \in V \) strategy \( p \ \sigma \) winning-strategy \( p \ \sigma \ \langle v \rangle \)
  \item \textbf{shows} \( v \in \text{winning-region} \ p \)
\end{itemize}

\begin{proof}

\end{proof}

\textbf{lemma} winning-region-in-V [simp]: winning-region \( p \subseteq V \) \( \langle \rangle \)

\begin{proof}

\end{proof}

\textbf{lemma} winning-region-deadends:
\begin{itemize}
  \item \textbf{assumes} \( v \in V \) \( p \) \( \text{deadend} \) \( v \)
  \item \textbf{shows} \( v \in \text{winning-region} \ p \) \( \ast \)
\end{itemize}

\begin{proof}

\end{proof}

\end{theory}

\end{context}

end — WellOrderedStrategies

\textbf{end}

30
9.1 Paths in Winning Regions

**Lemma (in vmc-path) paths-stay-in-winning-region:**

assumes \( \sigma': \text{strategy} p \sigma' \text{ winning-strategy} p \sigma' v0 \)
and \( \sigma: \bigwedge v. v \in \text{winning-region} p \implies \sigma' v = \sigma v \)
shows \( \text{set} \ P \subseteq \text{winning-region} p \)

**Proof**

**Lemma (in vmc-path) path-hits-winning-region-is-winning:**

assumes \( \sigma': \text{strategy} p \sigma' \bigwedge v. v \in \text{winning-region} p \implies \text{winning-strategy} p \sigma' v \)
and \( \sigma: \bigwedge v. v \in \text{winning-region} p \implies \sigma' v = \sigma v \)
and \( P: \text{set} \ P \cap \text{winning-region} p \neq \{\} \)
shows \( \text{winning-path} p P \)

**Proof**

9.2 Irrelevant Updates

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

**Lemma** winning-strategy-updates:

assumes \( \sigma: \text{strategy} p \sigma \text{ winning-strategy} p \sigma v0 \)
and \( v: v \notin \text{winning-region} p v \rightarrow w \)
shows \( \text{winning-strategy} p (\sigma(v := w)) v0 \)

**Proof**

9.3 Extending Winning Regions

**Lemma** winning-region-extends-VVp:

assumes \( v: v \in VV p v \rightarrow w \) and \( w: w \in \text{winning-region} p \)
shows \( v \in \text{winning-region} p \)

**Proof**

Unfortunately, we cannot prove the corresponding theorem \( \text{winning-region-extends-VVpstar} \) for \( VV p**-nodes yet. First, we need to show that there exists a uniform winning strategy on \( \text{winning-region} p \). We will prove \( \text{winning-region-extends-VVpstar} \) 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: $\bigwedge v. v \in S \implies \exists \sigma. \text{strategy } p \sigma \land \text{strategy-attracts-via } p \sigma v S W$
- **shows** $\exists \sigma. \text{strategy } p \sigma \land \text{strategy-attracts } p \sigma S W$

(\textit{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** $\exists \sigma. \text{strategy } p \sigma \land (\forall v \in \text{winning-region } p. \text{winning-strategy } p \sigma v)$

(\textit{proof})

10.3 Extending Winning Regions

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

**lemma** winning-region-extends-$VVp^{star}$:
- **assumes** $v. v \in VV p^{**}$ and $w. \bigwedge w. v \rightarrow w \implies w \in \text{winning-region } p$
- **shows** $v \in \text{winning-region } p$

(\textit{proof})

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

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

(\textit{proof})

end — context ParityGame

end

11 Attractor Strategies

**theory** AttractorStrategy

**imports**
- \textit{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** $\exists \sigma. \text{strategy } p \sigma \land \text{strategy-attracts-via } p \sigma v0$ (\textit{insert }v0\textit{ }S) \textit{W}$

(\textit{proof})
lemma strategy-attracts-extends-VVpstar:
  assumes $\sigma$: strategy-attracts $p \sigma S W$
  and $v0$: $v0 \notin VV p \sigma v0 \in$ directly-attracted $p S$
  shows strategy-attracts-via $p \sigma v0$ (insert $v0 S$) $W$
  ⟨proof⟩

lemma attractor-has-strategy-single:
  assumes $W \subseteq V$
  and $v0$-def: $v0 \in$ attractor $p W$ (is -$\in$ ?A)
  shows $\exists \sigma$. strategy $p \sigma \land$ strategy-attracts-via $p \sigma v0 ?A W$
  ⟨proof⟩

11.1 Existence

Prove that every attractor set has an attractor strategy.

theorem attractor-has-strategy:
  assumes $W \subseteq V$
  shows $\exists \sigma$. strategy $p \sigma \land$ strategy-attracts $p \sigma$ (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 $\omega \cdot V$, 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 $v \in V$
  and no-deadends: $\forall v \cdot v \in V \implies \neg$ deadend $v$
  and III: $\forall(G :: (a', b') ParityGame-scheme) v$. $\exists(v \cdot v \in V_G \implies \neg$ Digraph.deadend $G v$)
  $\implies \exists p \cdot v \in ParityGame.winning-region G p$
shows $\exists p. v \in \text{winning-region } p$
(proof)

12.2 Positional Determinacy without Deadends

theorem positional-strategy-exists-without-deadends:
  assumes $v \in V \land \forall v. v \in V \rightarrow \neg \text{deadend } v$
  shows $\exists p. v \in \text{winning-region } p$
(proof)

12.3 Positional Determinacy with Deadends

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

theorem positional-strategy-exists:
  assumes $v_0 \in V$
  shows $\exists p. v_0 \in \text{winning-region } p$
(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 $V$.

theorem partition-into-winning-regions:
  shows $V = \text{winning-region Even } \cup \text{winning-region Odd}$
  and $\text{winning-region Even } \cap \text{winning-region Odd } = \{}$
(proof)

end — context ParityGame

end

13 Defining the Attractor with inductive_set

theory AttractorInductive
import
  Main
  Attractor
begin

context ParityGame begin

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

13.1 $\text{attractor-inductive}$

The attractor set of a given set of nodes, defined inductively.
\textbf{inductive-set} \textit{attractor-inductive}: \textit{Player} \Rightarrow 'a \textit{set} \Rightarrow 'a \textit{set}

\textit{for p} :: \textit{Player} and \textit{W} :: 'a \textit{set} where
\begin{itemize}
    \item \textit{Base} [\text{intro!}]: \:\: v \in W \implies v \in \text{attractor-inductive} p W \\
    \item \textit{VVp}: \: [ \: v \in VV p; \: \exists w. \: v \rightarrow w \land w \in \text{attractor-inductive} p W \: ] \\
        \implies v \in \text{attractor-inductive} p W \\
    \item \textit{VVpstar}: \: [ \: v \in VV p**; \: \neg\text{deadend} v; \: \forall w. \: v \rightarrow w \implies w \in \text{attractor-inductive} p W \: ] \\
        \implies v \in \text{attractor-inductive} p W
\end{itemize}

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

\textbf{lemma} \textit{attractor-inductive-is-attractor}:
\begin{itemize}
    \item \textit{assumes} \: W \subseteq V \\
    \item \textit{shows} \: \text{attractor-inductive} p W = \text{attractor} p W
\end{itemize}
\textit{(proof)}

\textit{end}

\textit{end}

\section{14 Compatibility with the Graph Theory Package}

\begin{footnotesize}
\textbf{theory} \textit{Graph-TheoryCompatibility}  \\
\textbf{imports} \textit{ParityGame}  \\
\hspace{1em} \textit{Graph-Theory.Digraph}  \\
\hspace{1em} \textit{Graph-Theory.Digraph-Isomorphism}  \\
\textbf{begin}  \\
\textit{In this section, we show that our Digraph locale is compatible to the nomulti-digraph locale from the graph theory package from the Archive of Formal Proofs.}  \\
\textit{For this, we will define two functions converting between the different types and show that with these conversion functions the locales interpret each other. Together, this indicates that our definition of digraph is reasonable.}
\end{footnotesize}

\subsection{14.1 \textit{To Graph Theory}}

We can easily convert our graphs into \textit{pre-digraph} objects.

\begin{footnotesize}
\textbf{definition} \textit{to-pre-digraph} :: ('a, 'b) \textit{Graph-scheme} \Rightarrow ('a, 'a \times 'a) \textit{pre-digraph}  \\
\hspace{1em} \textit{where} \textit{to-pre-digraph} G \equiv ()  \\
\hspace{2em} \textit{pre-digraph.verts} = \textit{Graph.verts} G,  \\
\hspace{2em} \textit{pre-digraph.ares} = \textit{Graph.ares} G,  \\
\hspace{2em} \textit{tail} = \textit{fst},  \\
\hspace{2em} \textit{head} = \textit{snd}
\end{footnotesize}

With this conversion function, our \textit{Digraph} locale contains the locale \textit{nomulti-digraph} from the graph theory package.

\begin{footnotesize}
\textit{context} \textit{Digraph} \textit{begin}  \\
\textit{interpretation} \textit{is-nomulti-digraph}: \textit{nomulti-digraph} \textit{to-pre-digraph} G \textit{(proof)}  \\
\textit{end}
\end{footnotesize}
14.2 From Graph Theory

We can also convert in the other direction.

definition from-pre-digraph :: (a, b) pre-digraph ⇒ 'a Graph
  where from-pre-digraph G ≡ (Graph.verts = pre-digraph.verts G,
    Graph.arses = arcs-ends G)

context nomulti-digraph begin
interpretation is-Digraph: Digraph from-pre-digraph G (proof)
end

14.3 Isomorphisms

We also show that our conversion functions make sense. That is, we show that they are nearly inverses of each other. Unfortunately, from-pre-digraph irretrievably loses information about the arcs, and only keeps tail/head intact, so the best we can get for this case is that the back-and-forth converted graphs are isomorphic.

lemma graph-conversion-bij: G = from-pre-digraph (to-pre-digraph G) (proof)

lemma (in nomulti-digraph) graph-conversion-bij2: digraph-iso G (to-pre-digraph (from-pre-digraph G)) (proof)

end
References


