

# Positional Determinacy of Parity Games

Christoph Dittmann  
christoph.dittmann@tu-berlin.de

March 17, 2025

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 [5].

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Formal Introduction	4
1.2	Overview	4
1.3	Technical Aspects	5
<b>2</b>	<b>Auxiliary Lemmas for Coinductive Lists</b>	<b>5</b>
2.1	<i>lset</i>	5
2.2	<i>llength</i>	6
2.3	<i>ltake</i>	6
2.4	<i>ldropn</i>	6
2.5	<i>lfinite</i>	6
2.6	<i>lmap</i>	7
2.7	Notation	7
<b>3</b>	<b>Parity Games</b>	<b>7</b>
3.1	Basic definitions	7
3.2	Graphs	7
3.3	Valid Paths	8
3.4	Maximal Paths	9
3.5	Parity Games	10
3.6	Sets of Deadends	10
3.7	Subgames	10
3.8	Priorities Occurring Infinitely Often	11
3.9	Winning Condition	12
3.10	Valid Maximal Paths	13

<b>4</b>	<b>Positional Strategies</b>	<b>14</b>
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	<i>lset</i> Induction Schemas for Paths . . . . .	20
<b>5</b>	<b>Attracting Strategies</b>	<b>20</b>
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
<b>6</b>	<b>Attractor Sets</b>	<b>24</b>
6.1	<i>directly-attracted</i> . . . . .	24
6.2	<i>attractor-step</i> . . . . .	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
<b>7</b>	<b>Winning Strategies</b>	<b>26</b>
7.1	Deadends . . . . .	27
7.2	Extension Theorems . . . . .	27
<b>8</b>	<b>Well-Ordered Strategy</b>	<b>27</b>
8.1	Strategies on a Path . . . . .	29
8.2	Eventually One Strategy . . . . .	30
<b>9</b>	<b>Winning Regions</b>	<b>30</b>
9.1	Paths in Winning Regions . . . . .	31
9.2	Irrelevant Updates . . . . .	31
9.3	Extending Winning Regions . . . . .	31
<b>10</b>	<b>Uniform Strategies</b>	<b>31</b>
10.1	A Uniform Attractor Strategy . . . . .	32
10.2	A Uniform Winning Strategy . . . . .	32
10.3	Extending Winning Regions . . . . .	32
<b>11</b>	<b>Attractor Strategies</b>	<b>32</b>
11.1	Existence . . . . .	33
<b>12</b>	<b>Positional Determinacy of Parity Games</b>	<b>33</b>
12.1	Induction Step . . . . .	33

12.2	Positional Determinacy without Deadends . . . . .	34
12.3	Positional Determinacy with Deadends . . . . .	34
12.4	The Main Theorem: Positional Determinacy . . . . .	34
<b>13</b>	<b>Defining the Attractor with <code>inductive_set</code></b>	<b>34</b>
13.1	<i>attractor-inductive</i> . . . . .	35
<b>14</b>	<b>Compatibility with the Graph Theory Package</b>	<b>35</b>
14.1	To Graph Theory . . . . .	35
14.2	From Graph Theory . . . . .	36
14.3	Isomorphisms . . . . .	36
	<b>Bibliography</b>	<b>37</b>

# 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**  $vmc\text{-}path\ G\ P\ v_0\ p\ \sigma$  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  $lset\text{-}lnth$ :  $x \in lset\ xs \implies \exists n. lnth\ xs\ n = x$ 
  <proof>
```

```
lemma  $lset\text{-}lnth\text{-}member$ :  $\llbracket lset\ xs \subseteq A; enat\ n < llength\ xs \rrbracket \implies lnth\ xs\ n \in A$ 
  <proof>
```

```
lemma  $lset\text{-}nth\text{-}member\text{-}inf$ :  $\llbracket \neg lfinite\ xs; lset\ xs \subseteq A \rrbracket \implies lnth\ xs\ n \in A$ 
  <proof>
```

```
lemma  $lset\text{-}intersect\text{-}lnth$ :  $lset\ xs \cap A \neq \{\}$   $\implies \exists n. enat\ n < llength\ xs \wedge lnth\ xs\ n \in A$ 
  <proof>
```

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

```
lemma  $lfinite\text{-}lset$ :  $lfinite\ xs \implies \neg lnull\ xs \implies llast\ xs \in lset\ xs$ 
  <proof>
```

**lemma** *lset-subset*:  $\neg(\text{lset } xs \subseteq A) \implies \exists n. \text{enat } n < \text{llength } xs \wedge \text{lth } xs \ n \notin A$   
 ⟨proof⟩

## 2.2 llength

**lemma** *enat-Suc-ltl*:  
**assumes**  $\text{enat } (\text{Suc } n) < \text{llength } xs$   
**shows**  $\text{enat } n < \text{llength } (\text{ltl } xs)$   
 ⟨proof⟩

**lemma** *enat-ltl-Suc*:  $\text{enat } n < \text{llength } (\text{ltl } xs) \implies \text{enat } (\text{Suc } n) < \text{llength } xs$   
 ⟨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*:  $\text{enat } (\text{Suc } n) < \text{llength } xs \implies \text{enat } n < \text{llength } xs$   
 ⟨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]:  $n \leq m \implies \text{lset } (\text{ltake } n \ xs) \subseteq \text{lset } (\text{ltake } m \ xs)$   
 ⟨proof⟩

**lemma** *lset-ltake*:  $(\bigwedge m. m < n \implies \text{lnth } xs \ m \in A) \implies \text{lset } (\text{ltake } (\text{enat } n) \ xs) \subseteq A$   
 ⟨proof⟩

**lemma** *llength-ltake'*:  $\text{enat } n < \text{llength } xs \implies \text{llength } (\text{ltake } (\text{enat } n) \ xs) = \text{enat } n$   
 ⟨proof⟩

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

**lemma** *lset-ltake-ltl*:  $\text{lset } (\text{ltake } (\text{enat } n) \ (\text{ltl } xs)) \subseteq \text{lset } (\text{ltake } (\text{enat } (\text{Suc } n)) \ xs)$   
 ⟨proof⟩

## 2.4 ldropr

**lemma** *ltl-ldropr*:  $\llbracket \bigwedge xs. P \ xs \implies P \ (\text{ltl } xs); P \ xs \rrbracket \implies P \ (\text{ldropr } n \ xs)$   
 ⟨proof⟩

## 2.5 lfinite

**lemma** *lfinite-drop-set*:  $\text{lfinite } xs \implies \exists n. v \notin \text{lset } (\text{ldropr } n \ xs)$

*<proof>*

**lemma** *index-infinite-set*:

$\llbracket \neg \text{lfinite } x; \text{lnth } x \ m = y; \bigwedge i. \text{lnth } x \ i = y \implies (\exists m > i. \text{lnth } x \ m = y) \rrbracket \implies y \in \text{lset } (\text{ldropn } n \ x)$

*<proof>*

## 2.6 *lmap*

**lemma** *lnth-lmap-ldropn*:

$\text{enat } n < \text{llength } xs \implies \text{lnth } (\text{lmap } f \ (\text{ldropn } n \ xs)) \ 0 = \text{lnth } (\text{lmap } f \ xs) \ n$

*<proof>*

**lemma** *lnth-lmap-ldropn-Suc*:

$\text{enat } (\text{Suc } n) < \text{llength } xs \implies \text{lnth } (\text{lmap } f \ (\text{ldropn } n \ xs)) \ (\text{Suc } 0) = \text{lnth } (\text{lmap } f \ xs) \ (\text{Suc } n)$

*<proof>*

## 2.7 Notation

We introduce the notation  $\$$  to denote *lnth*.

**notation** *lnth* (**infix**  $\langle \$ \rangle$  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* × *'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* ( $\langle V_1 \rangle$ )

*arcs* :: *'a Edge set* ( $\langle E_1 \rangle$ )

**abbreviation** *is-arc* :: (*'a*, *'b*) *Graph-scheme*  $\Rightarrow$  *'a*  $\Rightarrow$  *'a*  $\Rightarrow$  *bool* (**infixl**  $\langle \rightarrow_1 \rangle$  60) **where**

$v \rightarrow_G w \equiv (v, w) \in E_G$

```

locale Digraph =
  fixes G (structure)
  assumes valid-edge-set:  $E \subseteq V \times V$ 
begin
lemma edges-are-in-V [intro]:  $v \rightarrow w \implies v \in V \ v \rightarrow w \implies w \in V$  <proof>

```

A node without successors is a *deadend*.

```

abbreviation deadend :: 'a  $\implies$  bool where deadend v  $\equiv \neg(\exists w \in V. v \rightarrow w)$ 

```

### 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  $\implies$  bool where
  valid-path-base: valid-path LNil
| valid-path-base':  $v \in V \implies$  valid-path (LCons v LNil)
| valid-path-cons:  $\llbracket v \in V; w \in V; v \rightarrow w; \text{valid-path } Ps; \neg \text{lnull } Ps; \text{lhs } Ps = w \rrbracket$ 
   $\implies$  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)  $\implies$  valid-path Ps
  <proof>

```

```

lemma valid-path-ltl: valid-path P  $\implies$  valid-path (ltl P)
  <proof>

```

```

lemma valid-path-drop: valid-path P  $\implies$  valid-path (ldropn n P)
  <proof>

```

```

lemma valid-path-in-V: assumes valid-path P shows lset P  $\subseteq$  V <proof>

```

```

lemma valid-path-finite-in-V:  $\llbracket \text{valid-path } P; \text{enat } n < \text{llength } P \rrbracket \implies P \$ n \in V$ 
  <proof>

```

```

lemma valid-path-edges': valid-path (LCons v (LCons w Ps))  $\implies v \rightarrow w$ 
  <proof>

```

```

lemma valid-path-edges:
  assumes valid-path P enat (Suc n) < llength P
  shows P $ n  $\rightarrow$  P $ Suc n
  <proof>

```

```

lemma valid-path-coinduct [consumes 1, case-names base step, coinduct pred: valid-path]:
  assumes major: Q P
  and base:  $\bigwedge v P. Q (LCons v LNil) \implies v \in V$ 
  and step:  $\bigwedge v w P. Q (LCons v (LCons w P)) \implies v \rightarrow w \wedge (Q (LCons w P) \vee \text{valid-path } (LCons w P))$ 
  shows valid-path P
  <proof>

```

```

lemma valid-path-no-deadends:
   $\llbracket \text{valid-path } P; \text{enat } (Suc i) < \text{llength } P \rrbracket \implies \neg \text{deadend } (P \$ i)$ 
  <proof>

```



**lemma** *valid-path-ends-on-deadend*:

$\llbracket \text{valid-path } P; \text{enat } i < \text{llength } P; \text{deadend } (P \$ i) \rrbracket \implies \text{enat } (\text{Suc } i) = \text{llength } P$   
<proof>

**lemma** *valid-path-prefix*:  $\llbracket \text{valid-path } P; \text{lprefix } P' P \rrbracket \implies \text{valid-path } P'$

<proof>

**lemma** *valid-path-lappend*:

**assumes** *valid-path*  $P$  *valid-path*  $P'$   $\llbracket \neg \text{lnull } P; \neg \text{lnull } P' \rrbracket \implies \text{llast } P \rightarrow \text{lhs } P'$

**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**  $G'$ : *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**

*maximal-path-base*: *maximal-path*  $LNil$

| *maximal-path-base'*: *deadend*  $v \implies \text{maximal-path } (LCons v LNil)$

| *maximal-path-cons*:  $\neg \text{lnull } Ps \implies \text{maximal-path } Ps \implies \text{maximal-path } (LCons v Ps)$

**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*  $P \implies \text{maximal-path } (\text{ltl } P)$

<proof>

**lemma** *maximal-drop*: *maximal-path*  $P \implies \text{maximal-path } (\text{ldropn } n P)$

<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'*:  $\llbracket \text{lfinite } P; \text{deadend } (\text{llast } P) \rrbracket \implies \text{maximal-path } P$

<proof>

**lemma** *infinite-path-is-maximal*:  $\llbracket \text{valid-path } P; \neg \text{lfinite } P \rrbracket \implies \text{maximal-path } P$

<proof>

**end** — locale *Digraph*

### 3.5 Parity Games

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

**datatype**  $Player = Even \mid Odd$

**abbreviation**  $other-player\ p \equiv (if\ p = Even\ then\ Odd\ else\ Even)$

**notation**  $other-player\ (\langle(-)**\rangle\ [1000]\ 1000)$

**lemma**  $other-other-player\ [simp]:\ p*** = p\ \langle proof \rangle$

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**  $'a\ ParityGame = 'a\ Graph +$   
 $player0 :: 'a\ set\ (\langle V_0 \rangle)$   
 $priority :: 'a \Rightarrow nat\ (\langle \omega \rangle)$

**locale**  $ParityGame = Digraph\ G\ \mathbf{for}\ G :: ('a, 'b)\ ParityGame-scheme\ (\mathbf{structure}) +$

**assumes**  $valid-player0-set: V_0 \subseteq V$

**and**  $priorities-finite: finite\ (\omega\ 'V)$

**begin**

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

**abbreviation**  $VV :: Player \Rightarrow 'a\ set\ \mathbf{where}\ VV\ p \equiv (if\ p = Even\ then\ V_0\ else\ V - V_0)$

**lemma**  $VVp-to-V\ [intro]:\ v \in VV\ p \Longrightarrow v \in V\ \langle proof \rangle$

**lemma**  $VV-impl1: v \in VV\ p \Longrightarrow v \notin VV\ p**\ \langle proof \rangle$

**lemma**  $VV-impl2: v \in VV\ p** \Longrightarrow v \notin VV\ p\ \langle proof \rangle$

**lemma**  $VV-equivalence\ [iff]:\ v \in V \Longrightarrow v \notin VV\ p \longleftrightarrow v \in VV\ p**\ \langle proof \rangle$

**lemma**  $VV-cases\ [consumes\ 1]:\ \llbracket v \in V ; v \in VV\ p \Longrightarrow P ; v \in VV\ p** \Longrightarrow P \rrbracket \Longrightarrow P\ \langle proof \rangle$

### 3.6 Sets of Deadends

**definition**  $deadends\ p \equiv \{v \in VV\ p.\ deadend\ v\}$

**lemma**  $deadends-in-V: deadends\ p \subseteq V\ \langle proof \rangle$

### 3.7 Subgames

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

**definition**  $subgame\ \mathbf{where}$

$subgame\ V' \equiv G\{$

$verts := V \cap V',$

$arcs := E \cap (V' \times V'),$

$player0 := V_0 \cap V' \}$

**lemma**  $subgame-V\ [simp]: V_{subgame\ V'} \subseteq V$

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

**and**  $subgame-\omega: \omega_{subgame\ V'} = \omega$

$\langle proof \rangle$

**lemma**

**assumes**  $V' \subseteq V$

**shows**  $subgame-V'\ [simp]: V_{subgame\ V'} = V'$

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

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

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

**lemma** *subgame-finite* [simp]:  $\text{finite } (\omega_{\text{subgame } V'} \text{ ' } V_{\text{subgame } V'})$  ⟨proof⟩

**lemma** *subgame- $\omega$ -subset* [simp]:  $\omega_{\text{subgame } V'} \text{ ' } V_{\text{subgame } V'} \subseteq \omega \text{ ' } V$   
 ⟨proof⟩

**lemma** *subgame-Digraph*:  $\text{Digraph } (\text{subgame } V')$   
 ⟨proof⟩

**lemma** *subgame-ParityGame*:  
**shows**  $\text{ParityGame } (\text{subgame } V')$   
 ⟨proof⟩

**lemma** *subgame-valid-path*:  
**assumes**  $P$ : *valid-path*  $P$  *lset*  $P \subseteq V'$   
**shows**  $\text{Digraph.valid-path } (\text{subgame } V') P$   
 ⟨proof⟩

**lemma** *subgame-maximal-path*:  
**assumes**  $V'$ :  $V' \subseteq V$  **and**  $P$ : *maximal-path*  $P$  *lset*  $P \subseteq V'$   
**shows**  $\text{Digraph.maximal-path } (\text{subgame } V') P$   
 ⟨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 **where**  
*path-inf-priorities*  $P \equiv \{k. \forall n. k \in \text{lset } (\text{ldropn } n (\text{lmap } \omega P))\}$

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*  $P$   $\neg$ *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**  $a$ :  $a \in \text{path-inf-priorities } P$   
**shows**  $\text{Min } (\omega \text{ ' } V) \leq a$   
 ⟨proof⟩

**lemma** *path-inf-priorities-LCons*:  
*path-inf-priorities*  $P = \text{path-inf-priorities } (\text{LCons } v P)$  (**is** ?A = ?B)  
 ⟨proof⟩

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

### 3.9 Winning Condition

Let  $G = (V, E, V_0, \omega)$  be a parity game. An infinite path  $v_0, v_1, \dots$  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* ::  $\text{Player} \Rightarrow 'a \text{ Path} \Rightarrow \text{bool}$  **where**  
*winning-path*  $p$   $P \equiv$   
 ( $\neg \text{lfinite } P \wedge (\exists a \in \text{path-inf-priorities } P.$   
 ( $\forall b \in \text{path-inf-priorities } P. a \leq b) \wedge \text{winning-priority } p a)$ )  
 $\vee (\neg \text{lnull } P \wedge \text{lfinite } P \wedge \text{llast } P \in VV \text{ } p^{**})$   
 $\vee (\text{lnull } P \wedge p = \text{Even})$

Every path has a unique winner.

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

**assumes** *valid-path*  $P$

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

⟨proof⟩

**lemma** *winning-path-ltl*:

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

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

⟨proof⟩

**corollary** *winning-path-drop*:

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

**shows**  $\text{winning-path } p (\text{ldropn } n P)$

⟨proof⟩

**corollary** *winning-path-drop-add*:

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

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

⟨proof⟩

**lemma** *winning-path-LCons*:

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

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

⟨proof⟩

**lemma** *winning-path-supergame*:

**assumes**  $\text{winning-path } p P$

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

**shows**  $\text{ParityGame.winning-path } G' p P$

⟨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]:  $\neg \text{lnull } P$ 
    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:  $P = \text{LCons } v0 \ (\text{ltl } P)$   $\langle \text{proof} \rangle$ 

lemma P-len [simp]:  $\text{enat } 0 < \text{llength } P$   $\langle \text{proof} \rangle$ 
lemma P-0 [simp]:  $P \ \$ \ 0 = v0$   $\langle \text{proof} \rangle$ 
lemma P-lnth-Suc:  $P \ \$ \ \text{Suc } n = \text{ltl } P \ \$ \ n$   $\langle \text{proof} \rangle$ 
lemma P-no-deadends:  $\text{enat } (\text{Suc } n) < \text{llength } P \implies \neg \text{deadend } (P \ \$ \ n)$ 
   $\langle \text{proof} \rangle$ 
lemma P-no-deadend-v0:  $\neg \text{lnull } (\text{ltl } P) \implies \neg \text{deadend } v0$ 
   $\langle \text{proof} \rangle$ 
lemma P-no-deadend-v0-llength:  $\text{enat } (\text{Suc } n) < \text{llength } P \implies \neg \text{deadend } v0$ 
   $\langle \text{proof} \rangle$ 
lemma P-ends-on-deadend:  $\llbracket \text{enat } n < \text{llength } P; \text{deadend } (P \ \$ \ n) \rrbracket \implies \text{enat } (\text{Suc } n) = \text{llength } P$ 
   $\langle \text{proof} \rangle$ 

lemma P-lnull-ltl-deadend-v0:  $\text{lnull } (\text{ltl } P) \implies \text{deadend } v0$ 
   $\langle \text{proof} \rangle$ 
lemma P-lnull-ltl-LCons:  $\text{lnull } (\text{ltl } P) \implies P = \text{LCons } v0 \ \text{LNil}$ 
   $\langle \text{proof} \rangle$ 
lemma P-deadend-v0-LCons:  $\text{deadend } v0 \implies P = \text{LCons } v0 \ \text{LNil}$ 
   $\langle \text{proof} \rangle$ 

lemma Ptl-valid [simp]: valid-path (ltl P)  $\langle \text{proof} \rangle$ 
lemma Ptl-maximal [simp]: maximal-path (ltl P)  $\langle \text{proof} \rangle$ 

lemma Pdrop-valid [simp]: valid-path (ldropn n P)  $\langle \text{proof} \rangle$ 
lemma Pdrop-maximal [simp]: maximal-path (ldropn n P)  $\langle \text{proof} \rangle$ 

lemma prefix-valid [simp]: valid-path (ltake n P)
   $\langle \text{proof} \rangle$ 

lemma extension-valid [simp]:  $v \rightarrow v0 \implies \text{valid-path } (\text{LCons } v \ P)$ 
   $\langle \text{proof} \rangle$ 
lemma extension-maximal [simp]: maximal-path (LCons v P)
   $\langle \text{proof} \rangle$ 
lemma lappend-maximal [simp]: maximal-path (lappend P' P)
   $\langle \text{proof} \rangle$ 

lemma v0-V [simp]:  $v0 \in V$   $\langle \text{proof} \rangle$ 

```

```

lemma v0-lset-P [simp]:  $v0 \in \text{lset } P$  <proof>
lemma v0-VV:  $v0 \in VV\ p \vee v0 \in VV\ p^{**}$  <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]:  $\text{lfinite } P \implies \text{deadend } (\text{llast } P)$ 
<proof>
lemma finite-llast-V [simp]:  $\text{lfinite } P \implies \text{llast } P \in V$ 
<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 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  $VV\ p$ .

```

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

```

```

lemma (in ParityGame) strategyI [intro]:
   $(\bigwedge v. \llbracket v \in VV\ p; \neg \text{deadend } v \rrbracket \implies v \rightarrow \sigma\ v) \implies \text{strategy } p\ \sigma$ 
<proof>

```

### 4.2 Strategy-Conforming Paths

If *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: path-conforms-with-strategy  $p\ \text{LNil}\ \sigma$ 
  | path-conforms-LCons-LNil: path-conforms-with-strategy  $p\ (\text{LCons } v\ \text{LNil})\ \sigma$ 

```

| *path-conforms-VVp*:  $\llbracket v \in VV\ p; w = \sigma\ v; \text{path-conforms-with-strategy}\ p\ (LCons\ w\ Ps)\ \sigma \rrbracket$   
 $\implies \text{path-conforms-with-strategy}\ p\ (LCons\ v\ (LCons\ w\ Ps))\ \sigma$   
| *path-conforms-VVpstar*:  $\llbracket v \notin VV\ p; \text{path-conforms-with-strategy}\ p\ Ps\ \sigma \rrbracket$   
 $\implies \text{path-conforms-with-strategy}\ p\ (LCons\ v\ Ps)\ \sigma$

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* = *vm-path* +  
**fixes** *p* *σ* **assumes** *P-conforms* [*simp*]: *path-conforms-with-strategy* *p* *P* *σ*

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

**locale** *vmc2-path* = *comp?*: *vmc-path* *G* *P* *v0* *p\*\** *σ'* + *vmc-path* *G* *P* *v0* *p* *σ*  
**for** *G* *P* *v0* *p* *σ* *σ'*

### 4.3 An Arbitrary Strategy

**context** *ParityGame* **begin**

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

**definition** *σ-arbitrary*  $\equiv \lambda v. \text{SOME } w. v \rightarrow w$

**lemma** *valid-arbitrary-strategy* [*simp*]: *strategy* *p* *σ-arbitrary*  $\langle \text{proof} \rangle$

### 4.4 Valid Strategies

**lemma** *valid-strategy-updates*:  $\llbracket \text{strategy}\ p\ \sigma; v0 \rightarrow w0 \rrbracket \implies \text{strategy}\ p\ (\sigma(v0 := w0))$   
 $\langle \text{proof} \rangle$

**lemma** *valid-strategy-updates-set*:  
**assumes** *strategy* *p* *σ*  $\wedge v. \llbracket v \in A; v \in VV\ p; \neg \text{deadend}\ v \rrbracket \implies v \rightarrow \sigma' v$   
**shows** *strategy* *p* (*override-on* *σ* *σ'* *A*)  
 $\langle \text{proof} \rangle$

**lemma** *valid-strategy-updates-set-strong*:  
**assumes** *strategy* *p* *σ* *strategy* *p* *σ'*  
**shows** *strategy* *p* (*override-on* *σ* *σ'* *A*)  
 $\langle \text{proof} \rangle$

**lemma** *subgame-strategy-stays-in-subgame*:  
**assumes** *σ*: *ParityGame.strategy* (*subgame* *V'*) *p* *σ*  
**and** *v*  $\in$  *ParityGame.VV* (*subgame* *V'*) *p*  $\neg \text{Digraph.deadend}$  (*subgame* *V'*) *v*  
**shows** *σ* *v*  $\in$  *V'*  
 $\langle \text{proof} \rangle$

**lemma** *valid-strategy-supergame*:  
**assumes** *σ*: *strategy* *p* *σ*  
**and** *σ'*: *ParityGame.strategy* (*subgame* *V'*) *p* *σ'*  
**and** *G'-no-deadends*:  $\bigwedge v. v \in V' \implies \neg \text{Digraph.deadend}$  (*subgame* *V'*) *v*

**shows** *strategy p (override-on  $\sigma$   $\sigma'$   $V'$ ) (is strategy p ? $\sigma$ )*  
 ⟨proof⟩

**lemma** *valid-strategy-in-V*:  $\llbracket \text{strategy } p \ \sigma; v \in VV \ p; \neg \text{deadend } v \rrbracket \implies \sigma \ v \in V$   
 ⟨proof⟩

**lemma** *valid-strategy-only-in-V*:  $\llbracket \text{strategy } p \ \sigma; \bigwedge v. v \in V \implies \sigma \ v = \sigma' \ v \rrbracket \implies \text{strategy } p \ \sigma'$   
 ⟨proof⟩

## 4.5 Conforming Strategies

**lemma** *path-conforms-with-strategy-ltl* [intro]:  
*path-conforms-with-strategy p P  $\sigma \implies \text{path-conforms-with-strategy } p \ (\text{ltl } P) \ \sigma$*   
 ⟨proof⟩

**lemma** *path-conforms-with-strategy-drop*:  
*path-conforms-with-strategy p P  $\sigma \implies \text{path-conforms-with-strategy } p \ (\text{ldropn } n \ P) \ \sigma$*   
 ⟨proof⟩

**lemma** *path-conforms-with-strategy-prefix*:  
*path-conforms-with-strategy p P  $\sigma \implies \text{lprefix } P' \ P \implies \text{path-conforms-with-strategy } p \ P' \ \sigma$*   
 ⟨proof⟩

**lemma** *path-conforms-with-strategy-irrelevant*:  
**assumes** *path-conforms-with-strategy p P  $\sigma \ v \notin \text{lset } P$*   
**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  $\sigma \ \text{deadend } v \vee v \notin VV \ p \ \text{valid-path } P$*   
**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  $\sigma \ \bigwedge v. v \in \text{lset } P \implies \sigma \ v = \sigma' \ v$*   
**shows** *path-conforms-with-strategy p P  $\sigma'$*   
 ⟨proof⟩

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

**lemma** *path-conforms-with-strategy-irrelevant-deadend'*:  
**assumes** *path-conforms-with-strategy p P ( $\sigma(v := w)$ )  $\text{deadend } v \vee v \notin VV \ p \ \text{valid-path } P$*   
**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**  
 $P$ : *lfinite*  $P$   $\neg$ *lnull*  $P$  *path-conforms-with-strategy*  $p$   $P$   $\sigma$   
**and**  $P'$ :  $\neg$ *lnull*  $P'$  *path-conforms-with-strategy*  $p$   $P'$   $\sigma$   
**and** *conforms*:  $llast$   $P \in VV$   $p \implies \sigma (llast P) = lhd P'$   
**shows** *path-conforms-with-strategy*  $p$  (*lappend*  $P P'$ )  $\sigma$   
 $\langle$ *proof* $\rangle$

**lemma** *path-conforms-with-strategy-VVpstar*:  
**assumes** *lset*  $P \subseteq VV$   $p^{**}$   
**shows** *path-conforms-with-strategy*  $p$   $P$   $\sigma$   
 $\langle$ *proof* $\rangle$

**lemma** *subgame-path-conforms-with-strategy*:  
**assumes**  $V'$ :  $V' \subseteq V$  **and**  $P$ : *path-conforms-with-strategy*  $p$   $P$   $\sigma$  *lset*  $P \subseteq V'$   
**shows** *ParityGame.path-conforms-with-strategy* (*subgame*  $V'$ )  $p$   $P$   $\sigma$   
 $\langle$ *proof* $\rangle$

**lemma** (**in** *vmc-path*) *subgame-path-vmc-path*:  
**assumes**  $V'$ :  $V' \subseteq V$  **and**  $P$ : *lset*  $P \subseteq V'$   
**shows** *vmc-path* (*subgame*  $V'$ )  $P$   $v0$   $p$   $\sigma$   
 $\langle$ *proof* $\rangle$

## 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*  $\Rightarrow$  'a *Strategy*  $\Rightarrow$  'a *Strategy*  $\Rightarrow$  'a  $\Rightarrow$  'a *Path*  
**where**

*greedy-conforming-path*  $p$   $\sigma$   $\sigma'$   $v0 =$   
 $LCons$   $v0$  (*if* *deadend*  $v0$   
then *LNil*  
else *if*  $v0 \in VV$   $p$   
then *greedy-conforming-path*  $p$   $\sigma$   $\sigma'$  ( $\sigma$   $v0$ )  
else *greedy-conforming-path*  $p$   $\sigma$   $\sigma'$  ( $\sigma'$   $v0$ ))

**private lemma** *greedy-path-LNil*: *greedy-conforming-path*  $p$   $\sigma$   $\sigma'$   $v0 \neq LNil$   
 $\langle$ *proof* $\rangle$  **lemma** *greedy-path-lhd*: *greedy-conforming-path*  $p$   $\sigma$   $\sigma'$   $v0 = LCons$   $v$   $P \implies v = v0$   
 $\langle$ *proof* $\rangle$  **lemma** *greedy-path-deadend-v0*: *greedy-conforming-path*  $p$   $\sigma$   $\sigma'$   $v0 = LCons$   $v$   $P \implies P = LNil \iff$  *deadend*  $v0$

$\langle$ *proof* $\rangle$  **corollary** *greedy-path-deadend-v*:  
*greedy-conforming-path*  $p$   $\sigma$   $\sigma'$   $v0 = LCons$   $v$   $P \implies P = LNil \iff$  *deadend*  $v$   
 $\langle$ *proof* $\rangle$

**corollary** *greedy-path-deadend-v'*: *greedy-conforming-path*  $p$   $\sigma$   $\sigma'$   $v0 = LCons$   $v$  *LNil*  $\implies$  *deadend*  $v$

⟨proof⟩ **lemma** *greedy-path-ltl*:  
**assumes** *greedy-conforming-path*  $p \ \sigma \ \sigma' \ v0 = LCons \ v \ P$   
**shows**  $P = LNil \vee P = \text{greedy-conforming-path } p \ \sigma \ \sigma' (\sigma \ v0) \vee P = \text{greedy-conforming-path } p \ \sigma \ \sigma' (\sigma' \ v0)$   
 ⟨proof⟩ **lemma** *greedy-path-ltl-ex*:  
**assumes** *greedy-conforming-path*  $p \ \sigma \ \sigma' \ v0 = LCons \ v \ P$   
**shows**  $P = LNil \vee (\exists v. P = \text{greedy-conforming-path } p \ \sigma \ \sigma' \ v)$   
 ⟨proof⟩ **lemma** *greedy-path-ltl-VVp*:  
**assumes** *greedy-conforming-path*  $p \ \sigma \ \sigma' \ v0 = LCons \ v0 \ P \ v0 \in VV \ p \ \neg \text{deadend } v0$   
**shows**  $\sigma \ v0 = \text{lhd } P$   
 ⟨proof⟩ **lemma** *greedy-path-ltl-VVpstar*:  
**assumes** *greedy-conforming-path*  $p \ \sigma \ \sigma' \ v0 = LCons \ v0 \ P \ v0 \in VV \ p^{**} \ \neg \text{deadend } v0$   
**shows**  $\sigma' \ v0 = \text{lhd } P$   
 ⟨proof⟩ **lemma** *greedy-conforming-path-properties*:  
**assumes**  $v0 \in V \ \text{strategy } p \ \sigma \ \text{strategy } p^{**} \ \sigma'$   
**shows**  
     *greedy-path-not-null*:  $\neg \text{lnull } (\text{greedy-conforming-path } p \ \sigma \ \sigma' \ v0)$   
     *greedy-path-v0*:  $\text{greedy-conforming-path } p \ \sigma \ \sigma' \ v0 \ \$ \ 0 = v0$   
     *greedy-path-valid*:  $\text{valid-path } (\text{greedy-conforming-path } p \ \sigma \ \sigma' \ v0)$   
     *greedy-path-maximal*:  $\text{maximal-path } (\text{greedy-conforming-path } p \ \sigma \ \sigma' \ v0)$   
     *greedy-path-conforms*:  $\text{path-conforms-with-strategy } p \ (\text{greedy-conforming-path } p \ \sigma \ \sigma' \ v0) \ \sigma$   
     *greedy-path-conforms'*:  $\text{path-conforms-with-strategy } p^{**} \ (\text{greedy-conforming-path } p \ \sigma \ \sigma' \ v0) \ \sigma'$   
 ⟨proof⟩

**corollary** *strategy-conforming-path-exists*:  
**assumes**  $v0 \in V \ \text{strategy } p \ \sigma \ \text{strategy } p^{**} \ \sigma'$   
**obtains**  $P$  **where** *vmc2-path*  $G \ P \ v0 \ p \ \sigma \ \sigma'$   
 ⟨proof⟩

**corollary** *strategy-conforming-path-exists-single*:  
**assumes**  $v0 \in V \ \text{strategy } p \ \sigma$   
**obtains**  $P$  **where** *vmc-path*  $G \ P \ v0 \ p \ \sigma$   
 ⟨proof⟩

end

end

## 4.7 Valid Maximal Conforming Paths

Now is the time to add some lemmas to the locale *vmc-path*.

**context** *vmc-path* **begin**  
**lemma** *Ptl-conforms* [*simp*]: *path-conforms-with-strategy*  $p \ (\text{ltl } P) \ \sigma$   
 ⟨proof⟩  
**lemma** *Pdrop-conforms* [*simp*]: *path-conforms-with-strategy*  $p \ (\text{ldropn } n \ P) \ \sigma$   
 ⟨proof⟩  
**lemma** *prefix-conforms* [*simp*]: *path-conforms-with-strategy*  $p \ (\text{ltake } n \ P) \ \sigma$   
 ⟨proof⟩  
**lemma** *extension-conforms* [*simp*]:  
 $(v' \in VV \ p \implies \sigma \ v' = v0) \implies \text{path-conforms-with-strategy } p \ (LCons \ v' \ P) \ \sigma$

*<proof>*

**lemma** *extension-valid-maximal-conforming*:

**assumes**  $v' \rightarrow v0$   $v' \in VV$   $p \implies \sigma$   $v' = v0$

**shows** *vmc-path*  $G$  (*LCons*  $v' P$ )  $v' p \sigma$

*<proof>*

**lemma** *vmc-path-ldropn*:

**assumes** *enat*  $n < \text{length } P$

**shows** *vmc-path*  $G$  (*ldropn*  $n P$ ) ( $P \$ n$ )  $p \sigma$

*<proof>*

**lemma** *conforms-to-another-strategy*:

*path-conforms-with-strategy*  $p P \sigma' \implies \text{vmc-path } G P v0 p \sigma'$

*<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*  $p P \sigma$

**shows** *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 *ltl P* compared to *vmc-path*.

**locale** *vmc-path-no-deadend* = *vmc-path* +

**assumes** *v0-no-deadend* [*simp*]:  $\neg \text{deadend } v0$

**begin**

**definition**  $w0 \equiv \text{lhd } (\text{ltl } P)$

**lemma** *Ptl-not-null* [*simp*]:  $\neg \text{null } (\text{ltl } P)$

*<proof>*

**lemma** *Ptl-LCons*:  $\text{ltl } P = \text{LCons } w0 (\text{ltl } (\text{ltl } P))$  *<proof>*

**lemma** *P-LCons'*:  $P = \text{LCons } v0 (\text{LCons } w0 (\text{ltl } (\text{ltl } P)))$  *<proof>*

**lemma** *v0-edge-w0* [*simp*]:  $v0 \rightarrow w0$  *<proof>*

**lemma** *Ptl-0*:  $\text{ltl } P \$ 0 = \text{lhd } (\text{ltl } P)$  *<proof>*

**lemma** *P-Suc-0*:  $P \$ \text{Suc } 0 = w0$  *<proof>*

**lemma** *Ptl-edge* [*simp*]:  $v0 \rightarrow \text{lhd } (\text{ltl } P)$  *<proof>*

**lemma** *v0-conforms*:  $v0 \in VV$   $p \implies \sigma$   $v0 = w0$

*<proof>*

**lemma** *w0-V* [*simp*]:  $w0 \in V$  *<proof>*

**lemma** *w0-lset-P* [*simp*]:  $w0 \in \text{lset } P$  *<proof>*

**lemma** *vmc-path-ltl* [*simp*]: *vmc-path*  $G$  (*ltl*  $P$ )  $w0 p \sigma$  *<proof>*

**end**

**context** *vmc-path* **begin**

**lemma** *vmc-path-lnull-ltl-no-deadend*:

$\neg \text{lnull } (\text{ltl } P) \implies \text{vmc-path-no-deadend } G P v0 p \sigma$   
<proof>

**lemma** *vmc-path-conforms*:

**assumes**  $\text{enat } (\text{Suc } n) < \text{llength } P P \$ n \in VV p$   
**shows**  $\sigma (P \$ n) = P \$ \text{Suc } n$   
<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*:  $\bigwedge P v0. \llbracket \text{vmc-path-no-deadend } G P v0 p \sigma; v0 \in S; Q P \rrbracket$   
 $\implies Q (\text{ltl } P) \wedge (\text{vmc-path-no-deadend.w0 } P) \in S$   
**shows**  $\text{lset } P \subseteq S$   
<proof>

$\llbracket ?Q P; v0 \in ?S; \bigwedge P v0. \llbracket \text{vmc-path-no-deadend } G P v0 p \sigma; v0 \in ?S; ?Q P \rrbracket \implies ?Q (\text{ltl } P) \wedge \text{vmc-path-no-deadend.w0 } P \in ?S \rrbracket \implies \text{lset } P \subseteq ?S$  without the Q predicate.

**corollary** *vmc-path-lset-induction-simple* [*case-names base step*]:

**assumes** *base*:  $v0 \in S$   
**and** *step*:  $\bigwedge P v0. \llbracket \text{vmc-path-no-deadend } G P v0 p \sigma; v0 \in S \rrbracket$   
 $\implies \text{vmc-path-no-deadend.w0 } P \in S$   
**shows**  $\text{lset } P \subseteq S$   
<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*:  $\bigwedge v. \llbracket v \in S; \neg \text{deadend } v; v \in VV p \rrbracket \implies \sigma v \in S \cup T$   
**and** *VVpstar*:  $\bigwedge v w. \llbracket v \in S; \neg \text{deadend } v; v \in VV p^{**}; v \rightarrow w \rrbracket \implies w \in S \cup T$   
**and** *v0*:  $v0 \in S$   
**and** *disjoint*:  $\text{lset } P \cap T = \{\}$   
**shows**  $\text{lset } P \subseteq S$   
<proof>

**end**

**end**

## 5 Attracting Strategies

**theory** *AttractingStrategy*

**imports**

*Main*

*Strategy*

**begin**

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

**lemma**  $visits\text{-}via\text{-}monotone: \llbracket visits\text{-}via\ P\ A\ W; A \subseteq A' \rrbracket \implies visits\text{-}via\ P\ A'\ W$   
(*proof*)

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

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

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

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

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

## 5.2 Attracting Strategy from a Single Node

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

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

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

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

**lemma** (in *vmc-path*) *strategy-attracts-via-lset*:

**assumes** *strategy-attracts-via*  $p \ \sigma \ v0 \ A \ W$

**shows**  $lset \ P \cap \ W \neq \{\}$

*<proof>*

**lemma** *strategy-attracts-via-v0*:

**assumes**  $\sigma$ : *strategy*  $p \ \sigma$  *strategy-attracts-via*  $p \ \sigma \ v0 \ A \ W$

**and**  $v0$ :  $v0 \in V$

**shows**  $v0 \in A \cup W$

*<proof>*

**corollary** *strategy-attracts-not-outside*:

$\llbracket v0 \in V - A - W; \text{strategy } p \ \sigma \rrbracket \implies \neg \text{strategy-attracts-via } p \ \sigma \ v0 \ A \ W$

*<proof>*

**lemma** *strategy-attracts-viaI* [*intro*]:

**assumes**  $\bigwedge P. \text{vmc-path } G \ P \ v0 \ p \ \sigma \implies \text{visits-via } P \ A \ W$

**shows** *strategy-attracts-via*  $p \ \sigma \ v0 \ A \ W$

*<proof>*

**lemma** *strategy-attracts-via-no-deadends*:

**assumes**  $v \in V \ v \in A - W$  *strategy-attracts-via*  $p \ \sigma \ v \ A \ W$

**shows**  $\neg \text{deadend } v$

*<proof>*

**lemma** *attractor-strategy-on-extends*:

$\llbracket \text{strategy-attracts-via } p \ \sigma \ v0 \ A \ W; A \subseteq A' \rrbracket \implies \text{strategy-attracts-via } p \ \sigma \ v0 \ A' \ W$

*<proof>*

**lemma** *strategy-attracts-via-trivial*:  $v0 \in W \implies \text{strategy-attracts-via } p \ \sigma \ v0 \ A \ W$

*<proof>*

**lemma** *strategy-attracts-via-successor*:

**assumes**  $\sigma$ : *strategy*  $p \ \sigma$  *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*  $p \ \sigma \ w0 \ A \ W$

*<proof>*

**lemma** *strategy-attracts-VVp*:

**assumes**  $\sigma$ : *strategy*  $p \ \sigma$  *strategy-attracts-via*  $p \ \sigma \ v0 \ A \ W$

**and**  $v$ :  $v0 \in A - W \ v0 \in VV \ p \ \neg \text{deadend } v0$

**shows**  $\sigma \ v0 \in A \cup W$

*<proof>*

**lemma** *strategy-attracts-VVpstar*:

**assumes** *strategy*  $p \ \sigma$  *strategy-attracts-via*  $p \ \sigma \ v0 \ A \ W$

**and**  $v0 \in A - W \ v0 \notin VV \ p \ w0 \in V - A - W$

**shows**  $\neg v0 \rightarrow w0$

*<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*  $\Rightarrow$  'a *set*  $\Rightarrow$  *bool* **where**  
*strategy-attracts*  $p \ \sigma \ A \ W \equiv \forall v0 \in A. \text{strategy-attracts-via } p \ \sigma \ v0 \ A \ W$

**lemma** (in *vmc-path*) *strategy-attractsE*:  
**assumes** *strategy-attracts*  $p \ \sigma \ A \ W \ v0 \in A$   
**shows** *visits-via*  $P \ A \ W$   
 $\langle$ *proof* $\rangle$

**lemma** *strategy-attractsI* [*intro*]:  
**assumes**  $\bigwedge P \ v. \llbracket v \in A; \text{vmc-path } G \ P \ v \ p \ \sigma \rrbracket \Longrightarrow \text{visits-via } P \ A \ W$   
**shows** *strategy-attracts*  $p \ \sigma \ A \ W$   
 $\langle$ *proof* $\rangle$

**lemma** (in *vmc-path*) *strategy-attracts-lset*:  
**assumes** *strategy-attracts*  $p \ \sigma \ A \ W \ v0 \in A$   
**shows** *lset*  $P \cap W \neq \{\}$   
 $\langle$ *proof* $\rangle$

**lemma** *strategy-attracts-empty* [*simp*]: *strategy-attracts*  $p \ \sigma \ \{\} \ W \ \langle$ *proof* $\rangle$

**lemma** *strategy-attracts-invalid-path*:  
**assumes**  $P: P = LCons \ v \ (LCons \ w \ P')$   $v \in A - W \ w \notin A \cup W$   
**shows**  $\neg \text{visits-via } P \ A \ W$  (is  $\neg ?A$ )  
 $\langle$ *proof* $\rangle$

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  $\sigma$  avoids this edge. All other cases give a contradiction.

**lemma** *strategy-attracts-does-not-leave*:  
**assumes**  $\sigma: \text{strategy-attracts } p \ \sigma \ A \ W \ \text{strategy } p \ \sigma$   
**and**  $v: v \rightarrow w \ v \in A - W \ w \notin A \cup W$   
**shows**  $v \in VV \ p \wedge \sigma \ v \neq w$   
 $\langle$ *proof* $\rangle$

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-override*:  
**assumes** *strategy-attracts*  $p \ \sigma \ A \ W \ \text{strategy } p \ \sigma \ \text{strategy } p \ \sigma'$   
**shows** *strategy-attracts*  $p \ (\text{override-on } \sigma' \ \sigma \ (A - W)) \ A \ W$   
 $\langle$ *proof* $\rangle$

**lemma** *strategy-attracts-trivial* [*simp*]: *strategy-attracts*  $p \ \sigma \ W \ W$   
 $\langle$ *proof* $\rangle$

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

**lemma** (in *vmc-path*) *attracted-path*:  
**assumes**  $W \subseteq V$

**and**  $\sigma$ : *strategy-attracts*  $p \ \sigma \ A \ W$   
**and** *P-hits-A*:  $lset \ P \cap \ A \neq \ \{\}$   
**shows**  $lset \ P \cap \ W \neq \ \{\}$   
*<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*]:  $directly-attracted \ p \ \{\} = \ \{\}$



**and** *directly-attracted-V-empty* [simp]: *directly-attracted*  $p$   $V = \{\}$   
**and** *directly-attracted-bounded-by-V* [simp]: *directly-attracted*  $p$   $W \subseteq V$   
**and** *directly-attracted-contains-no-deadends* [elim]:  $v \in \text{directly-attracted } p \ W \implies \neg \text{deadend } v$   
 ⟨proof⟩

## 6.2 attractor-step

**lemma** *attractor-step-empty*: *attractor-step*  $p$   $\{\}$   $\{\}$  =  $\{\}$   
**and** *attractor-step-bounded-by-V*:  $\llbracket W \subseteq V; S \subseteq V \rrbracket \implies \text{attractor-step } p \ W \ S \subseteq V$   
 ⟨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*  $p$   $W = \text{attractor-step } p \ W$  (*attractor*  $p$   $W$ )  
 ⟨proof⟩

**lemma** *attractor-lowerbound*: *attractor-step*  $p$   $W \ S \subseteq S \implies \text{attractor } p \ W \subseteq S$   
 ⟨proof⟩

**lemma** *attractor-set-non-empty*:  $W \neq \{\} \implies \text{attractor } p \ W \neq \{\}$   
**and** *attractor-set-base*:  $W \subseteq \text{attractor } p \ W$   
 ⟨proof⟩

**lemma** *attractor-in-V*:  $W \subseteq V \implies \text{attractor } p \ W \subseteq V$   
 ⟨proof⟩

## 6.4 Attractor Set Extensions

**lemma** *attractor-set-VVp*:  
**assumes**  $v \in VV \ p \ v \rightarrow w \ w \in \text{attractor } p \ W$   
**shows**  $v \in \text{attractor } p \ W$   
 ⟨proof⟩

**lemma** *attractor-set-VVpstar*:  
**assumes**  $\neg \text{deadend } v \ \wedge \ w. \ v \rightarrow w \implies w \in \text{attractor } p \ W$   
**shows**  $v \in \text{attractor } p \ W$   
 ⟨proof⟩

## 6.5 Removing an Attractor

**lemma** *removing-attractor-induces-no-deadends*:  
**assumes**  $v \in S - \text{attractor } p \ W \ v \rightarrow w \ w \in S \ \wedge \ w. \ \llbracket v \in VV \ p^{**}; v \rightarrow w \rrbracket \implies w \in S$   
**shows**  $\exists w \in S - \text{attractor } p \ W. \ v \rightarrow w$   
 ⟨proof⟩

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

**lemma** *subgame-without-deadends*:  
**assumes**  $V'\text{-def}: V' = V - \text{attractor } p \ (\text{deadends } p^{**}) - \text{attractor } p^{**} \ (\text{deadends } p^{****})$   
**(is**  $V' = V - ?A - ?B$ )  
**and**  $v: v \in V_{\text{subgame } V'}$

**shows**  $\neg \text{Digraph.deadend (subgame } V') v$   
 $\langle \text{proof} \rangle$

## 6.6 Attractor Set Induction

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

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

**and** *union*:  $\bigwedge M. \forall S \in M. S \subseteq V \wedge P S \implies P (\bigcup M)$

**shows**  $P (\text{attractor } p W)$

$\langle \text{proof} \rangle$

**end** — context `ParityGame`

**end**

## 7 Winning Strategies

**theory** *WinningStrategy*

**imports**

*Main*

*Strategy*

**begin**

**context** *ParityGame* **begin**

Here we define winning strategies.

A strategy is winning for player  $p$  from  $v0$  if every maximal  $\sigma$ -path starting in  $v0$  is winning.

**definition** *winning-strategy* :: *Player*  $\Rightarrow$  'a *Strategy*  $\Rightarrow$  'a  $\Rightarrow$  *bool* **where**

*winning-strategy*  $p \sigma v0 \equiv \forall P. \text{vmc-path } G P v0 p \sigma \longrightarrow \text{winning-path } p P$

**lemma** *winning-strategyI* [*intro*]:

**assumes**  $\bigwedge P. \text{vmc-path } G P v0 p \sigma \implies \text{winning-path } p P$

**shows** *winning-strategy*  $p \sigma v0$

$\langle \text{proof} \rangle$

**lemma** (**in** *vmc-path*) *paths-hits-winning-strategy-is-winning*:

**assumes**  $\sigma: \text{winning-strategy } p \sigma v$

**and**  $v: v \in \text{lset } P$

**shows** *winning-path*  $p P$

$\langle \text{proof} \rangle$

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

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

**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**  $v \in VV$   $p$  deadend  $v$  strategy  $p$   $\sigma$   
**shows**  $\neg$ winning-strategy  $p$   $\sigma$   $v$   
*<proof>*

**lemma** *winning-strategy-on-deadends*:  
**assumes**  $v \in VV$   $p$  deadend  $v$  strategy  $p$   $\sigma$   
**shows** winning-strategy  $p^{**}$   $\sigma$   $v$   
*<proof>*

## 7.2 Extension Theorems

**lemma** *strategy-extends-VVp*:  
**assumes**  $v0$ :  $v0 \in VV$   $p$   $\neg$ 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**  $v0$ :  $v0 \in VV$   $p^{**}$   $v0 \rightarrow w0$   
**and**  $\sigma$ : winning-strategy  $p$   $\sigma$   $v0$   
**shows** winning-strategy  $p$   $\sigma$   $w0$   
*<proof>*

**lemma** *strategy-extends-backwards-VVpstar*:  
**assumes**  $v0$ :  $v0 \in VV$   $p^{**}$   
**and**  $\sigma$ : strategy  $p$   $\sigma$   $\bigwedge w. v0 \rightarrow w \implies$  winning-strategy  $p$   $\sigma$   $w$   
**shows** winning-strategy  $p$   $\sigma$   $v0$   
*<proof>*

**lemma** *strategy-extends-backwards-VVp*:  
**assumes**  $v0$ :  $v0 \in VV$   $p$   $\sigma$   $v0 = w$   $v0 \rightarrow w$   
**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*

```

imports
  Main
  Strategy
begin

```

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  $S :: 'a \text{ set}$ 
    and  $p :: \text{Player}$ 
    — The set of good strategies on a node  $v$ 
    and  $good :: 'a \Rightarrow 'a \text{ Strategy set}$ 
    and  $r :: ('a \text{ Strategy} \times 'a \text{ Strategy}) \text{ set}$ 
  assumes  $S\text{-}V: S \subseteq V$ 
    —  $r$  is a wellorder on the set of all strategies which are good somewhere.
    and  $r\text{-wo}: \text{well-order-on } \{\sigma. \exists v \in S. \sigma \in good\ v\} \ r$ 
    — Every node has a good strategy.
    and  $good\text{-ex}: \bigwedge v. v \in S \Longrightarrow \exists \sigma. \sigma \in good\ v$ 
    — good strategies are well-formed strategies.
    and  $good\text{-strategies}: \bigwedge v \sigma. \sigma \in good\ v \Longrightarrow \text{strategy } p\ \sigma$ 
    — A good strategy on  $v$  is also good on possible successors of  $v$ .
    and  $strategies\text{-continue}: \bigwedge v\ w\ \sigma. \llbracket v \in S; v \rightarrow w; v \in VV\ p \Longrightarrow \sigma\ v = w; \sigma \in good\ v \rrbracket \Longrightarrow \sigma \in$ 
  good\ w
begin

```

The set of all strategies which are good somewhere.

**abbreviation**  $Strategies \equiv \{\sigma. \exists v \in S. \sigma \in good\ v\}$

**definition** *minimal-good-strategy* **where**

*minimal-good-strategy*  $v\ \sigma \equiv \sigma \in good\ v \wedge (\forall \sigma'. (\sigma', \sigma) \in r - Id \longrightarrow \sigma' \notin good\ v)$

**unbundle** *no binomial-syntax*

Among the good strategies on  $v$ , choose the minimum.

**definition** *choose* **where**

*choose*  $v \equiv \text{THE } \sigma. \text{minimal-good-strategy } v\ \sigma$

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*  $\equiv \text{override-on } \sigma\text{-arbitrary } (\lambda v. \text{choose } v\ v)\ S$

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

**lemma**  $r\text{-refl}$  [*simp*]: *refl-on Strategies*  $r$

*<proof>*

**lemma** *r-total [simp]: total-on Strategies r*  
⟨proof⟩

**lemma** *r-trans [simp]: trans r*  
⟨proof⟩

**lemma** *r-wf [simp]: wf (r - Id)*  
⟨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 ∈ good v*  
    **and** *choose-minimal:  $\bigwedge \sigma'. (\sigma', \text{choose } v) \in r - \text{Id} \implies \sigma' \notin \text{good } v$*   
    **and** *choose-strategy: strategy p (choose v)*  
⟨proof⟩

**corollary** *choose-in-Strategies:  $v \in S \implies \text{choose } v \in \text{Strategies}$*  ⟨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  $\equiv \text{lmap choose}$*

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

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

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

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

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

**lemma** *path-strategies-eventually-constant*:  
**assumes**  $\neg lfinite\ P\ lset\ P \subseteq S\ valid\text{-}path\ P\ path\text{-}conforms\text{-}with\text{-}strategy\ p\ P\ well\text{-}ordered\text{-}strategy$   
**shows**  $\exists n. \forall m \geq n. path\text{-}strategies\ P\ \$\ n = path\text{-}strategies\ P\ \$\ m$   
 $\langle proof \rangle$

## 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- $\sigma$ -map- $n$* :  
**assumes**  $lset\ P \subseteq S\ valid\text{-}path\ P\ path\text{-}conforms\text{-}with\text{-}strategy\ p\ P\ well\text{-}ordered\text{-}strategy$   
**shows**  $\exists n. path\text{-}conforms\text{-}with\text{-}strategy\ p\ (ldropn\ n\ P)\ (path\text{-}strategies\ P\ \$\ n)$   
 $\langle proof \rangle$

**end** — WellOrderedStrategies

**end**

## 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*  $p \equiv \{ v \in V. \exists \sigma. strategy\ p\ \sigma \wedge winning\text{-}strategy\ p\ \sigma\ v \}$

**lemma** *winning-regionI* [*intro*]:  
**assumes**  $v \in V\ strategy\ p\ \sigma\ winning\text{-}strategy\ p\ \sigma\ v$   
**shows**  $v \in winning\text{-}region\ p$   
 $\langle proof \rangle$

**lemma** *winning-region-in-V* [*simp*]:  $winning\text{-}region\ p \subseteq V$   $\langle proof \rangle$

**lemma** *winning-region-deadends*:  
**assumes**  $v \in VV\ p\ deadend\ v$   
**shows**  $v \in winning\text{-}region\ p^{**}$

*<proof>*

## 9.1 Paths in Winning Regions

**lemma** (in *vmc-path*) *paths-stay-in-winning-region*:  
 **assumes**  $\sigma'$ : *strategy*  $p$   $\sigma'$  *winning-strategy*  $p$   $\sigma' v0$   
 **and**  $\sigma$ :  $\bigwedge v. v \in \text{winning-region } p \implies \sigma' v = \sigma v$   
 **shows**  $\text{lset } P \subseteq \text{winning-region } p$   
*<proof>*

**lemma** (in *vmc-path*) *path-hits-winning-region-is-winning*:  
 **assumes**  $\sigma'$ : *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**  $P$ :  $\text{lset } P \cap \text{winning-region } p \neq \{\}$   
 **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**  $\sigma$ : *strategy*  $p$   $\sigma$  *winning-strategy*  $p$   $\sigma v0$   
 **and**  $v$ :  $v \notin \text{winning-region } p$   $v \rightarrow w$   
 **shows** *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 *winning-region-extends-VVpstar* 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-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*:  $\bigwedge v. v \in S \implies \exists \sigma. \text{strategy } p \ \sigma \wedge \text{strategy-attracts-via } p \ \sigma \ v \ S \ W$

**shows**  $\exists \sigma. \text{strategy } p \ \sigma \wedge \text{strategy-attracts } p \ \sigma \ S \ W$

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

*<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**  $w: \bigwedge w. v \rightarrow w \implies w \in \text{winning-region } p$

**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**  $v \in V - \text{winning-region } p \neg \text{deadend } v$

**shows**  $\exists w \in V - \text{winning-region } p. v \rightarrow w$

*<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**  $\sigma: \text{strategy } p \ \sigma \text{ strategy-attracts } p \ \sigma \ S \ W$

**and**  $v0: v0 \in VV \ p \ v0 \in \text{directly-attracted } p \ S \ v0 \notin S$



**shows**  $\exists \sigma. \text{strategy } p \ \sigma \wedge \text{strategy-attracts-via } p \ \sigma \ v0 \ (\text{insert } v0 \ S) \ W$   
 $\langle \text{proof} \rangle$

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

**lemma** *attractor-has-strategy-single*:  
**assumes**  $W \subseteq V$   
**and**  $v0\text{-def}: v0 \in \text{attractor } p \ W \ (\text{is } - \in ?A)$   
**shows**  $\exists \sigma. \text{strategy } p \ \sigma \wedge \text{strategy-attracts-via } p \ \sigma \ v0 \ ?A \ W$   
 $\langle \text{proof} \rangle$

## 11.1 Existence

Prove that every attractor set has an attractor strategy.

**theorem** *attractor-has-strategy*:  
**assumes**  $W \subseteq V$   
**shows**  $\exists \sigma. \text{strategy } p \ \sigma \wedge \text{strategy-attracts } p \ \sigma \ (\text{attractor } p \ W) \ W$   
 $\langle \text{proof} \rangle$

**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  $\omega \text{ ' } V$ , 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*:  $\bigwedge v. v \in V \implies \neg \text{deadend } v$   
**and** *IH*:  $\bigwedge (G :: ('a, 'b) \text{ParityGame-scheme}) v.$   
 $\llbracket \text{card } (\omega_G \text{ ' } V_G) < \text{card } (\omega \text{ ' } V); v \in V_G;$   
 $\text{ParityGame } G;$

```

     $\bigwedge v. v \in V_G \implies \neg \text{Digraph.deadend } G \ v \ ]$ 
     $\implies \exists p. v \in \text{ParityGame.winning-region } G \ p$ 
shows  $\exists p. v \in \text{winning-region } p$ 
<proof>

```

## 12.2 Positional Determinacy without Deadends

```

theorem positional-strategy-exists-without-deadends:
  assumes  $v \in V \ \bigwedge v. v \in V \implies \neg \text{deadend } v$ 
  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  $v0 \in V$ 
  shows  $\exists p. v0 \in \text{winning-region } p$ 
  <proof>

```

## 12.4 The Main Theorem: Positional Determinacy

Prove the main theorem: The winning regions of player EVEN and ODD are a partition of the set of nodes  $V$ .

```

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

```

**end** — context ParityGame

**end**

## 13 Defining the Attractor with inductive\_set

```

theory AttractorInductive
imports
  Main
  Attractor
begin

```

**context** *ParityGame* **begin**

In section 6 we defined *attractor* manually via *lfp*. We can also define it with `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 *attractor-inductive*

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

```
inductive-set attractor-inductive :: Player  $\Rightarrow$  'a set  $\Rightarrow$  'a set
  for p :: Player and W :: 'a set where
    Base [intro!]: v  $\in$  W  $\implies$  v  $\in$  attractor-inductive p W
  | VVp:  $\llbracket$  v  $\in$  VV p;  $\exists$  w. v  $\rightarrow$  w  $\wedge$  w  $\in$  attractor-inductive p W  $\rrbracket$ 
     $\implies$  v  $\in$  attractor-inductive p W
  | VVpstar:  $\llbracket$  v  $\in$  VV p**;  $\neg$ deadend v;  $\forall$  w. v  $\rightarrow$  w  $\longrightarrow$  w  $\in$  attractor-inductive p W  $\rrbracket$ 
     $\implies$  v  $\in$  attractor-inductive p W
```

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

```
lemma attractor-inductive-is-attractor:
  assumes W  $\subseteq$  V
  shows attractor-inductive p W = attractor p W
<proof>
```

**end**

**end**

## 14 Compatibility with the Graph Theory Package

```
theory Graph-TheoryCompatibility
imports
  ParityGame
  Graph-Theory.Digraph
  Graph-Theory.Digraph-Isomorphism
begin
```

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.

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.

### 14.1 To Graph Theory

We can easily convert our graphs into *pre-digraph* objects.

```
definition to-pre-digraph :: ('a, 'b) Graph-scheme  $\Rightarrow$  ('a, 'a  $\times$  'a) pre-digraph
  where to-pre-digraph G  $\equiv$  ( $\lfloor$ 
    pre-digraph.verts = Graph.verts G,
    pre-digraph.arcs = Graph.arcs G,
    tail = fst,
    head = snd
   $\rfloor$ )
```

With this conversion function, our *Digraph* locale contains the locale *nomulti-digraph* from the graph theory package.

```

context Digraph begin
interpretation is-nomulti-digraph: nomulti-digraph to-pre-digraph G  $\langle$ proof $\rangle$ 
end

```

## 14.2 From Graph Theory

We can also convert in the other direction.

```

definition from-pre-digraph :: ('a, 'b) pre-digraph  $\Rightarrow$  'a Graph
  where from-pre-digraph G  $\equiv$   $\langle$ 
    Graph.verts = pre-digraph.verts G,
    Graph.arcs = arcs-ends G
   $\rangle$ 

```

```

context nomulti-digraph begin
interpretation is-Digraph: Digraph from-pre-digraph G  $\langle$ proof $\rangle$ 
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)
   $\langle$ proof $\rangle$ 

```

```

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

```

```

end

```

## References

- [1] Julian Bradfield and Colin Stirling. Modal mu-calculi. In Patrick Blackburn, Johan Van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, pages 721 – 756. Elsevier, 2007.
- [2] Stephan Kreutzer. Logik, Spiele und Automaten. <http://logic.las.tu-berlin.de/Teaching/index.html>, 2015. Lecture notes for a master’s course on mathematical logic and games at Technische Universität Berlin (in German).
- [3] Ralf Küsters. Memoryless determinacy of parity games. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*, pages 95–106. Springer, 2001.
- [4] Andreas Lochbihler. Coinductive. *Archive of Formal Proofs*, February 2010. <http://isa-afp.org/entries/Coinductive.shtml>, Formal proof development.
- [5] Wiesaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.