

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

1	Introduction	4
1.1	Formal Introduction	4
1.2	Overview	4
1.3	Technical Aspects	5
2	Auxiliary Lemmas for Coinductive Lists	5
2.1	<i>lset</i>	5
2.2	<i>llength</i>	6
2.3	<i>ltake</i>	6
2.4	<i>ldropn</i>	7
2.5	<i>lfinite</i>	7
2.6	<i>lmap</i>	8
2.7	Notation	8
3	Parity Games	8
3.1	Basic definitions	8
3.2	Graphs	9
3.3	Valid Paths	9
3.4	Maximal Paths	11
3.5	Parity Games	12
3.6	Sets of Deadends	13
3.7	Subgames	13
3.8	Priorities Occurring Infinitely Often	15
3.9	Winning Condition	16
3.10	Valid Maximal Paths	18

4	Positional Strategies	20
4.1	Definitions	20
4.2	Strategy-Conforming Paths	20
4.3	An Arbitrary Strategy	20
4.4	Valid Strategies	21
4.5	Conforming Strategies	22
4.6	Greedy Conforming Path	25
4.7	Valid Maximal Conforming Paths	28
4.8	Valid Maximal Conforming Paths with One Edge	29
4.9	<i>lset</i> Induction Schemas for Paths	30
5	Attracting Strategies	31
5.1	Paths Visiting a Set	31
5.2	Attracting Strategy from a Single Node	32
5.3	Attracting strategy from a set of nodes	35
6	Attractor Sets	38
6.1	<i>directly-attracted</i>	38
6.2	<i>attractor-step</i>	38
6.3	Basic Properties of an Attractor	39
6.4	Attractor Set Extensions	39
6.5	Removing an Attractor	39
6.6	Attractor Set Induction	40
7	Winning Strategies	41
7.1	Deadends	42
7.2	Extension Theorems	43
8	Well-Ordered Strategy	44
8.1	Strategies on a Path	46
8.2	Eventually One Strategy	48
9	Winning Regions	49
9.1	Paths in Winning Regions	50
9.2	Irrelevant Updates	51
9.3	Extending Winning Regions	51
10	Uniform Strategies	52
10.1	A Uniform Attractor Strategy	52
10.2	A Uniform Winning Strategy	54
10.3	Extending Winning Regions	56
11	Attractor Strategies	56
11.1	Existence	58
12	Positional Determinacy of Parity Games	58
12.1	Induction Step	59

12.2	Positional Determinacy without Deadends	67
12.3	Positional Determinacy with Deadends	67
12.4	The Main Theorem: Positional Determinacy	70
13	Defining the Attractor with <code>inductive_set</code>	70
13.1	<i>attractor-inductive</i>	70
14	Compatibility with the Graph Theory Package	72
14.1	To Graph Theory	72
14.2	From Graph Theory	72
14.3	Isomorphisms	73
	Bibliography	75

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 “ σ -traps”. Using σ -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 $nat \Rightarrow 'a \text{ option}$ 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 $UNIV :: 'a \text{ set}$ but rather a subset of $UNIV :: 'a \text{ set}$, 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 σ for player p .

2 Auxiliary Lemmas for Coinductive Lists

Some lemmas to allow better reasoning with coinductive lists.

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

2.1 lset

```
lemma lset-lnth:  $x \in \text{lset } xs \implies \exists n. \text{lnth } xs\ n = x$ 
by (induct rule: llist.set-induct, meson lnth-0, meson lnth-Suc-LCons)
```

```
lemma lset-lnth-member:  $\llbracket \text{lset } xs \subseteq A; \text{enat } n < \text{llength } xs \rrbracket \implies \text{lnth } xs\ n \in A$ 
using contra-subsetD[of lset xs A] in-lset-conv-lnth[of - xs] by blast
```

```
lemma lset-nth-member-inf:  $\llbracket \neg \text{lfinit} e xs; \text{lset } xs \subseteq A \rrbracket \implies \text{lnth } xs\ n \in A$ 
by (metis contra-subsetD inf-llist-lnth lset-inf-llist rangeI)
```

```
lemma lset-intersect-lnth:  $\text{lset } xs \cap A \neq \{\} \implies \exists n. \text{enat } n < \text{llength } xs \wedge \text{lnth } xs\ n \in A$ 
by (metis disjoint-iff-not-equal in-lset-conv-lnth)
```

```
lemma lset-ltake-Suc:
  assumes  $\neg \text{lnull } xs \text{ lnth } xs\ 0 = x \text{ lset } (\text{ltake } (\text{enat } n) (\text{ttl } xs)) \subseteq A$ 
  shows  $\text{lset } (\text{ltake } (\text{enat } (\text{Suc } n)) xs) \subseteq \text{insert } x\ A$ 
proof–
  have  $\text{lset } (\text{ltake } (\text{eSuc } (\text{enat } n)) (\text{LCons } x (\text{ttl } xs))) \subseteq \text{insert } x\ A$ 
    using assms(3) by auto
  moreover from assms(1,2) have  $\text{LCons } x (\text{ttl } xs) = xs$ 
    by (metis lnth-0 ttl-simps(2) not-tnull-conv)
```

ultimately show *?thesis* **by** (*simp add: eSuc-enat*)
qed

lemma *lfinite-lset*: *lfinite xs \implies \neg lnull xs \implies llast xs \in lset xs*
proof (*induct rule: lfinite-induct*)
case (*LCons xs*)
show *?case* **proof** (*cases*)
assume *: *\neg lnull (ltl xs)*
hence *llast (ltl xs) \in lset (ltl xs)* **using** *LCons.hyps(3)* **by** *blast*
hence *llast (ltl xs) \in lset xs* **by** (*simp add: in-lset-ltlD*)
thus *?thesis* **by** (*metis * LCons.prem1 lhd-LCons-ltl llast-LCons2*)
qed (*metis LCons.prem1 lhd-LCons-ltl llast-LCons llist.set-sel(1)*)
qed *simp*

lemma *lset-subset*: *\neg (lset xs \subseteq A) \implies $\exists n$. enat n < llength xs \wedge lnth xs n \notin A*
by (*metis in-lset-conv-lnth subsetI*)

2.2 llength

lemma *enat-Suc-ltl*:
assumes *enat (Suc n) < llength xs*
shows *enat n < llength (ltl xs)*
proof—
from *assms* **have** *eSuc (enat n) < llength xs* **by** (*simp add: eSuc-enat*)
hence *enat n < epred (llength xs)* **using** *eSuc-le-iff ileI1* **by** *fastforce*
thus *?thesis* **by** (*simp add: epred-llength*)
qed

lemma *enat-ltl-Suc*: *enat n < llength (ltl xs) \implies enat (Suc n) < llength xs*
by (*metis eSuc-enat ldrop-ltl leD leI lnull-ldrop*)

lemma *infinite-small-llength* [*intro*]: *\neg lfinite xs \implies enat n < llength xs*
using *enat-iless lfinite-conv-llength-enat neg-iff* **by** *blast*

lemma *lnull-0-llength*: *\neg lnull xs \implies enat 0 < llength xs*
using *zero-enat-def* **by** *auto*

lemma *Suc-llength*: *enat (Suc n) < llength xs \implies enat n < llength xs*
using *dual-order.strict-trans enat-ord-simps(2)* **by** *blast*

2.3 ltake

lemma *ltake-lnth*: *ltake n xs = ltake n ys \implies enat m < n \implies lnth xs m = lnth ys m*
by (*metis lnth-ltake*)

lemma *lset-ltake-prefix* [*simp*]: *n \leq m \implies lset (ltake n xs) \subseteq lset (ltake m xs)*
by (*simp add: lprefix-lsetD*)

lemma *lset-ltake*: *($\bigwedge m$. m < n \implies lnth xs m \in A) \implies lset (ltake (enat n) xs) \subseteq A*
proof (*induct n arbitrary: xs*)
case 0
have *ltake (enat 0) xs = LNil* **by** (*simp add: zero-enat-def*)

thus ?case by simp
 next
 case (Suc n)
 show ?case proof (cases)
 assume $xs \neq LNil$
 then obtain $x\ xs'$ where $xs: xs = LCons\ x\ xs'$ by (meson neq-LNil-conv)
 { fix m assume $m < n$
 hence $Suc\ m < Suc\ n$ by simp
 hence $lnth\ xs\ (Suc\ m) \in A$ using $Suc.premis$ by presburger
 hence $lnth\ xs'\ m \in A$ using xs by simp
 }
 hence $lset\ (ltake\ (enat\ n)\ xs') \subseteq A$ using $Suc.hyps$ by blast
 moreover have $ltake\ (enat\ (Suc\ n))\ xs = LCons\ x\ (ltake\ (enat\ n)\ xs')$
 using $xs\ ltake-eSuc-LCons[of\ -\ x\ xs']$ by (metis (no-types) eSuc-enat)
 moreover have $x \in A$ using $Suc.premis\ xs$ by force
 ultimately show ?thesis by simp
 qed simp
 qed

lemma $llength-ltake'$: $enat\ n < llength\ xs \implies llength\ (ltake\ (enat\ n)\ xs) = enat\ n$
 by (metis $llength-ltake\ min.strict-order-iff$)

lemma $llast-ltake$:
 assumes $enat\ (Suc\ n) < llength\ xs$
 shows $llast\ (ltake\ (enat\ (Suc\ n))\ xs) = lnth\ xs\ n$ (is $llast\ ?A = -$)
 unfolding $llast-def$ using $llength-ltake'[OF\ assms]$ by (auto simp add: $lnth-ltake$)

lemma $lset-ltake-ltl$: $lset\ (ltake\ (enat\ n)\ (ltl\ xs)) \subseteq lset\ (ltake\ (enat\ (Suc\ n))\ xs)$
 proof (cases)
 assume $\neg lnull\ xs$
 then obtain $v0$ where $xs = LCons\ v0\ (ltl\ xs)$ by (metis $lhd-LCons-ltl$)
 hence $ltake\ (eSuc\ (enat\ n))\ xs = LCons\ v0\ (ltake\ (enat\ n)\ (ltl\ xs))$
 by (metis $ltake-eSuc-LCons$)
 hence $lset\ (ltake\ (enat\ (Suc\ n))\ xs) = lset\ (LCons\ v0\ (ltake\ (enat\ n)\ (ltl\ xs)))$
 by (simp add: $eSuc-enat$)
 thus ?thesis using $lset-LCons[of\ v0\ ltake\ (enat\ n)\ (ltl\ xs)]$ by blast
 qed (simp add: $lnull-def$)

2.4 $ldropn$

lemma $ltl-ldrop$: $\llbracket \bigwedge xs. P\ xs \implies P\ (ltl\ xs); P\ xs \rrbracket \implies P\ (ldropn\ n\ xs)$
 unfolding $ldropn-def$ by (induct n) simp-all

2.5 $lfinite$

lemma $lfinite-drop-set$: $lfinite\ xs \implies \exists n. v \notin lset\ (ldrop\ n\ xs)$
 by (metis $ldrop-inf\ lmember-code(1)\ lset-lmember$)

lemma $index-infinite-set$:
 $\llbracket \neg lfinite\ x; lnth\ x\ m = y; \bigwedge i. lnth\ x\ i = y \implies (\exists m > i. lnth\ x\ m = y) \rrbracket \implies y \in lset\ (ldropn\ n\ x)$
 proof (induct n arbitrary: $x\ m$)

```

  case 0 thus ?case using lset-nth-member-inf by auto
next
case (Suc n)
obtain a xs where x: x = LCons a xs by (meson Suc.prem1) lnull-imp-lfinite not-llnull-conv
obtain j where j: j > m lnth x j = y using Suc.prem2,3 by blast
have lnth xs (j - 1) = y by (metis lnth-LCons' j(1,2) not-less0 x)
moreover {
  fix i assume lnth xs i = y
  hence lnth x (Suc i) = y by (simp add: x)
  hence  $\exists j > i. \text{lnth } xs \ j = y$  by (metis Suc.prem3 Suc-lessE lnth-Suc-LCons x)
}
ultimately show ?case using Suc.hyps Suc.prem1 x by auto
qed

```

2.6 lmap

lemma *lnth-lmap-ldropn*:

```

  enat n < llength xs  $\implies$  lnth (lmap f (ldropn n xs)) 0 = lnth (lmap f xs) n
  by (simp add: lhd-ldropn lnth-0-conv-lhd)

```

lemma *lnth-lmap-ldropn-Suc*:

```

  enat (Suc n) < llength xs  $\implies$  lnth (lmap f (ldropn n xs)) (Suc 0) = lnth (lmap f xs) (Suc n)
  by (metis (no-types, lifting) Suc-llength ldropn-ltl leD llist.map-disc-iff lnth-lmap-ldropn
      lnth-ltl lnull-ldropn ltl-ldropn ltl-lmap)

```

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 $'a \text{ Edge} = 'a \times 'a$

A path is a possibly infinite list of nodes.

type-synonym $'a \text{ Path} = 'a \text{ llist}$

3.2 Graphs

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

```

record 'a Graph =
  verts :: 'a set (⟨V1⟩)
  arcs :: 'a Edge set (⟨E1⟩)
abbreviation is-arc :: ('a, 'b) Graph-scheme ⇒ 'a ⇒ 'a ⇒ bool (infixl ⟨→1⟩ 60) where
  v →G w ≡ (v,w) ∈ EG

locale Digraph =
  fixes G (structure)
  assumes valid-edge-set: E ⊆ V × V
begin
lemma edges-are-in-V [intro]: v→w ⇒ v ∈ V v→w ⇒ w ∈ V using valid-edge-set by blast+

```

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; ¬lnull Ps; lhd Ps = w ]
  ⇒ valid-path (LCons v Ps)

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

lemma valid-path-ltl': valid-path (LCons v Ps) ⇒ valid-path Ps
  using valid-path.simps by blast
lemma valid-path-ltl: valid-path P ⇒ valid-path (ltl P)
  by (metis llist.exhaust-sel ltl-simps(1) valid-path-ltl')
lemma valid-path-drop: valid-path P ⇒ valid-path (ldropn n P)
  by (simp add: valid-path-ltl ltl-ldrop)

lemma valid-path-in-V: assumes valid-path P shows lset P ⊆ V proof
  fix x assume x ∈ lset P thus x ∈ V
  using assms by (induct rule: llist.set-induct) (auto intro: valid-path.cases)
qed
lemma valid-path-finite-in-V: [ valid-path P; enat n < llength P ] ⇒ P $ n ∈ V
  using valid-path-in-V lset-lnth-member by blast

lemma valid-path-edges': valid-path (LCons v (LCons w Ps)) ⇒ v→w
  using valid-path.cases by fastforce
lemma valid-path-edges:
  assumes valid-path P enat (Suc n) < llength P
  shows P $ n → P $ Suc n
proof–

```

define P' **where** $P' = \text{ldropn } n \ P$
have $\text{enat } n < \text{llength } P$ **using** $\text{assms}(2) \ \text{enat-ord-simps}(2) \ \text{less-trans}$ **by** blast
hence $P' \$ 0 = P \$ n$ **by** $(\text{simp add: } P'\text{-def})$
moreover have $P' \$ \text{Suc } 0 = P \$ \text{Suc } n$
by $(\text{metis One-nat-def } P'\text{-def } \text{Suc-eq-plus1 add commute assms}(2) \ \text{lnth-ldropn})$
ultimately have $\exists Ps. P' = \text{LCons } (P \$ n) (\text{LCons } (P \$ \text{Suc } n) \ Ps)$
by $(\text{metis } P'\text{-def } \langle \text{enat } n < \text{llength } P \rangle \ \text{assms}(2) \ \text{ldropn-Suc-conv-ldropn})$
moreover have $\text{valid-path } P'$ **by** $(\text{simp add: } P'\text{-def assms}(1) \ \text{valid-path-drop})$
ultimately show $?thesis$ **using** valid-path-edges' **by** blast
qed

lemma $\text{valid-path-coinduct}$ $[\text{consumes } 1, \text{ case-names base step, coinduct pred: valid-path}]$:
assumes $\text{major: } Q \ P$
and $\text{base: } \bigwedge v \ P. \ Q \ (\text{LCons } v \ \text{LNil}) \implies v \in V$
and $\text{step: } \bigwedge v \ w \ P. \ Q \ (\text{LCons } v \ (\text{LCons } w \ P)) \implies v \rightarrow w \wedge (Q \ (\text{LCons } w \ P) \vee \text{valid-path } (\text{LCons } w \ P))$
shows $\text{valid-path } P$
using major proof $(\text{coinduction arbitrary: } P)$
case valid-path
{ **assume** $P \neq \text{LNil}$ $\neg(\exists v. P = \text{LCons } v \ \text{LNil} \wedge v \in V)$
then obtain $v \ w \ P'$ **where** $P = \text{LCons } v \ (\text{LCons } w \ P')$
using $\text{neq-LNil-conv base valid-path by metis}$
hence $?case$ **using** $\text{step valid-path by auto}$
}
thus $?case$ **by** blast
qed

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

lemma $\text{valid-path-ends-on-deadend}$:
 $\llbracket \text{valid-path } P; \text{enat } i < \text{llength } P; \text{deadend } (P \$ i) \rrbracket \implies \text{enat } (\text{Suc } i) = \text{llength } P$
using $\text{valid-path-no-deadends by (metis enat-iless enat-ord-simps}(2) \ \text{neq-iff not-less-eq})$

lemma valid-path-prefix : $\llbracket \text{valid-path } P; \text{lprefix } P' \ P \rrbracket \implies \text{valid-path } P'$
proof $(\text{coinduction arbitrary: } P' \ P)$
case $(\text{step } v \ w \ P'' \ P' \ P)$
then obtain Ps **where** $Ps: \text{LCons } v \ (\text{LCons } w \ Ps) = P$ **by** $(\text{metis LCons-lprefix-conv})$
hence $\text{valid-path } (\text{LCons } w \ Ps)$ **using** $\text{valid-path-ltl' step}(2)$ **by** blast
moreover have $\text{lprefix } (\text{LCons } w \ P'') \ (\text{LCons } w \ Ps)$ **using** $Ps \ \text{step}(1,3)$ **by** auto
ultimately show $?case$ **using** $Ps \ \text{step}(2) \ \text{valid-path-edges' by blast}$
qed $(\text{metis LCons-lprefix-conv valid-path-cons-simp})$

lemma $\text{valid-path-lappend}$:
assumes $\text{valid-path } P \ \text{valid-path } P' \llbracket \neg \text{lnull } P; \neg \text{lnull } P' \rrbracket \implies \text{llast } P \rightarrow \text{lhs } P'$
shows $\text{valid-path } (\text{lappend } P \ P')$
proof (cases, cases)
assume $\neg \text{lnull } P \ \neg \text{lnull } P'$
thus $?thesis$ **using** assms proof $(\text{coinduction arbitrary: } P' \ P)$
case $(\text{step } v \ w \ P'' \ P' \ P)$
show $?case$ **proof** (cases)

```

assume lnull (ltl P)
thus ?case using step(1,2,3,5,6)
  by (metis lhd-LCons lhd-LCons-ltl lhd-lappend llast-singleton
      llist.collapse(1) ltl-lappend ltl-simps(2))
next
assume  $\neg$ lnull (ltl P)
moreover have ltl (lappend P P') = lappend (ltl P) P' using step(2) by simp
ultimately show ?case using step
  by (metis (no-types, lifting)
      lhd-LCons lhd-LCons-ltl lhd-lappend llast-LCons ltl-simps(2)
      valid-path-edges' valid-path-ltl)
qed
qed (metis llist.disc(1) lnull-lappend ltl-lappend ltl-simps(2))
qed (simp-all add: assms(1,2) lappend-lnull1 lappend-lnull2)

```

A valid path is still valid in a supergame.

```

lemma valid-path-supergame:
  assumes valid-path P and G': Digraph G' V  $\subseteq$  VG' E  $\subseteq$  EG'
  shows Digraph.valid-path G' P
using  $\langle$ valid-path P $\rangle$  proof (coinduction arbitrary: P
  rule: Digraph.valid-path-coinduct[OF G'(1), case-names base step])
  case base thus ?case using G'(2) valid-path-cons-simp by auto
qed (meson G'(3) subset-eq valid-path-edges' valid-path-ltl')

```

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  $\implies$  maximal-path (LCons v LNil)
| maximal-path-cons:  $\neg$ lnull Ps  $\implies$  maximal-path Ps  $\implies$  maximal-path (LCons v Ps)

```

```

lemma maximal-no-deadend: maximal-path (LCons v Ps)  $\implies$   $\neg$ deadend v  $\implies$   $\neg$ lnull Ps
  by (metis lhd-LCons llist.distinct(1) ltl-simps(2) maximal-path.simps)

```

```

lemma maximal-ltl: maximal-path P  $\implies$  maximal-path (ltl P)
  by (metis ltl-simps(1) ltl-simps(2) maximal-path.simps)

```

```

lemma maximal-drop: maximal-path P  $\implies$  maximal-path (ldropn n P)
  by (simp add: maximal-ltl ltl-ldrop)

```

```

lemma maximal-path-lappend:
  assumes  $\neg$ lnull P' maximal-path P'
  shows maximal-path (lappend P P')
proof (cases)
  assume  $\neg$ lnull P
  thus ?thesis using assms proof (coinduction arbitrary: P' P rule: maximal-path.coinduct)
    case (maximal-path P' P)
    let ?P = lappend P P'
    show ?case proof (cases ?P = LNil  $\vee$  ( $\exists v$ . ?P = LCons v LNil  $\wedge$  deadend v))
      case False
      then obtain Ps v where P: ?P = LCons v Ps by (meson neq-LNil-conv)
      hence Ps = lappend (ltl P) P' by (simp add: lappend-ltl maximal-path(1))

```

```

    hence  $\exists Ps1 P'. Ps = lappend Ps1 P' \wedge \neg lnull P' \wedge maximal\_path P'$ 
    using maximal-path(2) maximal-path(3) by auto
    thus ?thesis using P lappend-lnull1 by fastforce
qed blast
qed
qed (simp add: assms(2) lappend-lnull1[of P P'])

lemma maximal-ends-on-deadend:
  assumes maximal-path P lfinite P  $\neg lnull P$ 
  shows deadend (llast P)
proof-
  from  $\langle lfinite P \rangle \langle \neg lnull P \rangle$  obtain n where n: llength P = enat (Suc n)
  by (metis enat-ord-simps(2) gr0-implies-Suc lfinite-llength-enat lnull-0-llength)
  define P' where P' = ldropn n P
  hence maximal-path P' using assms(1) maximal-drop by blast
  thus ?thesis proof (cases rule: maximal-path.cases)
  case (maximal-path-base' v)
  hence deadend (llast P') unfolding P'-def by simp
  thus ?thesis unfolding P'-def using llast-ldropn[of n P] n
  by (metis P'-def ldropn-eq-LConsD local.maximal-path-base'(1))
next
  case (maximal-path-cons P'' v)
  hence ldropn (Suc n) P = P'' unfolding P'-def by (metis ldrop-eSuc-ltl ltl-ldropn ltl-simps(2))
  thus ?thesis using n maximal-path-cons(2) by auto
qed (simp add: P'-def n ldropn-eq-LNil)
qed

lemma maximal-ends-on-deadend':  $\llbracket lfinite P; deadend (llast P) \rrbracket \implies maximal\_path P$ 
proof (coinduction arbitrary: P rule: maximal-path.coinduct)
  case (maximal-path P)
  show ?case proof (cases)
  assume P  $\neq LNil$ 
  then obtain v P' where P': P = LCons v P' by (meson neq-LNil-conv)
  show ?thesis proof (cases)
  assume P' = LNil thus ?thesis using P' maximal-path(2) by auto
  qed (metis P' lfinite-LCons llast-LCons llist.collapse(1) maximal-path(1,2))
qed simp
qed

lemma infinite-path-is-maximal:  $\llbracket valid\_path P; \neg lfinite P \rrbracket \implies maximal\_path P$ 
by (coinduction arbitrary: P rule: maximal-path.coinduct)
(cases rule: valid-path.cases, auto)

```

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 (if p = Even then Odd else Even)

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

lemma *other-other-player* [simp]: $p**** = p$ **using** *Player.exhaust* **by** *auto*

A parity game is tuple (V, E, V_0, ω) , where (V, E) is a graph, $V_0 \subseteq V$ and ω 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* **for** $G :: ('a, 'b)$ *ParityGame-scheme* (**structure**) +
 assumes *valid-player0-set*: $V_0 \subseteq V$
 and *priorities-finite*: *finite* (ω ' V)
begin

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

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

lemma *VVp-to-V* [intro]: $v \in VV\ p \implies v \in V$ **using** *valid-player0-set* **by** (*cases p*) *auto*

lemma *VV-impl1*: $v \in VV\ p \implies v \notin VV\ p**$ **by** *auto*

lemma *VV-impl2*: $v \in VV\ p** \implies v \notin VV\ p$ **by** *auto*

lemma *VV-equivalence* [iff]: $v \in V \implies v \notin VV\ p \longleftrightarrow v \in VV\ p**$ **by** *auto*

lemma *VV-cases* [consumes 1]: $\llbracket v \in V ; v \in VV\ p \implies P ; v \in VV\ p** \implies P \rrbracket \implies P$ **by** *auto*

3.6 Sets of Deadends

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

lemma *deadends-in-V*: *deadends p* $\subseteq V$ **unfolding** *deadends-def* **by** *blast*

3.7 Subgames

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

definition *subgame* **where**

subgame V' $\equiv G \langle$
 verts := $V \cap V'$,
 arcs := $E \cap (V' \times V')$,
 player0 := $V_0 \cap V'$ \rangle

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

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

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

unfolding *subgame-def* **by** *simp-all*

lemma

assumes $V' \subseteq V$

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

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

unfolding *subgame-def* **using** *assms* **by** *auto*

lemma *subgame-VV* [simp]: $ParityGame.VV\ (subgame\ V')\ p = V' \cap VV\ p$ **proof**–

have $ParityGame.VV\ (subgame\ V')\ Even = V' \cap VV\ Even$ **unfolding** *subgame-def* **by** *auto*

moreover have $ParityGame.VV\ (subgame\ V')\ Odd = V' \cap VV\ Odd$ **proof**–

```

    have  $V' \cap V - (V0 \cap V') = V' \cap V \cap (V - V0)$  by blast
    thus ?thesis unfolding subgame-def by auto
  qed
  ultimately show ?thesis by simp
qed
corollary subgame-VV-subset [simp]: ParityGame.VV (subgame  $V'$ )  $p \subseteq VV$   $p$  by simp

lemma subgame-finite [simp]: finite ( $\omega_{\text{subgame } V'} \text{ ' } V_{\text{subgame } V'}$ ) proof–
  have finite ( $\omega \text{ ' } V_{\text{subgame } V'}$ ) using subgame-V priorities-finite
  by (meson finite-subset image-mono)
  thus ?thesis by (simp add: subgame-def)
qed

lemma subgame- $\omega$ -subset [simp]:  $\omega_{\text{subgame } V'} \text{ ' } V_{\text{subgame } V'} \subseteq \omega \text{ ' } V$ 
  by (simp add: image-mono subgame- $\omega$ )

lemma subgame-Digraph: Digraph (subgame  $V'$ )
  by (unfold-locales) (auto simp add: subgame-def)

lemma subgame-ParityGame:
  shows ParityGame (subgame  $V'$ )
proof (unfold-locales)
  show  $E_{\text{subgame } V'} \subseteq V_{\text{subgame } V'} \times V_{\text{subgame } V'}$ 
    using subgame-Digraph[unfolded Digraph-def] .
  show  $V0_{\text{subgame } V'} \subseteq V_{\text{subgame } V'}$  unfolding subgame-def using valid-player0-set by auto
  show finite ( $\omega_{\text{subgame } V'} \text{ ' } V_{\text{subgame } V'}$ ) by simp
qed

lemma subgame-valid-path:
  assumes  $P$ : valid-path  $P$  lset  $P \subseteq V'$ 
  shows Digraph.valid-path (subgame  $V'$ )  $P$ 
proof–
  have lset  $P \subseteq V$  using  $P(1)$  valid-path-in-V by blast
  hence lset  $P \subseteq V_{\text{subgame } V'}$  unfolding subgame-def using  $P(2)$  by auto
  with  $P(1)$  show ?thesis
proof (coinduction arbitrary: P)
  rule: Digraph.valid-path.coinduct[OF subgame-Digraph, case-names IH]
  case IH
  thus ?case proof (cases rule: valid-path.cases)
    case (valid-path-cons  $v$   $w$   $Ps$ )
    moreover hence  $v \in V_{\text{subgame } V'}$   $w \in V_{\text{subgame } V'}$  using  $IH(2)$  by auto
    moreover hence  $v \rightarrow_{\text{subgame } V'} w$  using local.valid-path-cons(4) subgame-def by auto
    moreover have valid-path  $Ps$  using  $IH(1)$  valid-path-ltl' local.valid-path-cons(1) by blast
    ultimately show ?thesis using  $IH(2)$  by auto
  qed auto
qed
qed

lemma subgame-maximal-path:
  assumes  $V'$ :  $V' \subseteq V$  and  $P$ : maximal-path  $P$  lset  $P \subseteq V'$ 

```

```

  shows Digraph.maximal-path (subgame V') P
proof-
  have lset P  $\subseteq V_{\text{subgame } V'}$  unfolding subgame-def using P(2) V' by auto
  with P(1) V' show ?thesis
  by (coinduction arbitrary: P rule: Digraph.maximal-path.coinduct[OF subgame-Digraph])
    (cases rule: maximal-path.cases, auto)
qed

```

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 $\equiv \{k. \forall n. k \in \text{lset } (\text{ldropn } n \text{ } (\text{lmap } \omega \text{ } P))\}$

Because ω 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–

Define a map from indices to priorities on the path.

```

define f where f i =  $\omega (P \$ i)$  for i
have range f  $\subseteq \omega ' V$  unfolding f-def
  using valid-path-in-V[OF P(1)] lset-nth-member-inf[OF P(2)]
  by blast
hence finite (range f)
  using priorities-finite finite-subset by blast
then obtain n0 where n0:  $\neg(\text{finite } \{n. f n = f n0\})$ 
  using pigeonhole-infinite[of UNIV f] by auto
define k where k = f n0

```

The priority *k* occurs infinitely often.

```

have lmap  $\omega P \$ n0 = k$  unfolding f-def k-def
  using assms(2) by (simp add: infinite-small-llength)
moreover {
  fix n assume lmap  $\omega P \$ n = k$ 
  have  $\exists n' > n. f n' = k$  unfolding k-def using n0 infinite-nat-iff-unbounded by auto
  hence  $\exists n' > n. \text{lmap } \omega P \$ n' = k$  unfolding f-def
    using assms(2) by (simp add: infinite-small-llength)
}
ultimately have  $\forall n. k \in \text{lset } (\text{ldropn } n \text{ } (\text{lmap } \omega P))$ 
  using index-infinite-set[of lmap  $\omega P n0 k$  ] P(2) lfinite-lmap
  by blast
thus ?thesis unfolding path-inf-priorities-def by blast
qed

```

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–
have $a \in \text{lset } (\text{ldropn } 0 \text{ (lmap } \omega \text{ } P))$ **using** a **unfolding** *path-inf-priorities-def* **by** *blast*
hence $a \in \omega \text{ ' lset } P$ **by** *simp*
thus *?thesis* **using** P *valid-path-in-V priorities-finite Min-le* **by** *blast*
qed

lemma *path-inf-priorities-LCons*:

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

proof

show $?A \subseteq ?B$ **proof**
fix a **assume** $a \in ?A$
hence $\forall n. a \in \text{lset } (\text{ldropn } n \text{ (lmap } \omega \text{ (} LCons \text{ } v \text{ } P)))$
unfolding *path-inf-priorities-def*
using *in-lset-ltlD[of a]* **by** *(simp add: ltl-ldropn)*
thus $a \in ?B$ **unfolding** *path-inf-priorities-def* **by** *blast*
qed

next

show $?B \subseteq ?A$ **proof**
fix a **assume** $a \in ?B$
hence $\forall n. a \in \text{lset } (\text{ldropn } (Suc \text{ } n) \text{ (lmap } \omega \text{ (} LCons \text{ } v \text{ } P)))$
unfolding *path-inf-priorities-def* **by** *blast*
thus $a \in ?A$ **unfolding** *path-inf-priorities-def* **by** *simp*
qed

qed

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

by *(metis llist.exhaust ltl-simps path-inf-priorities-LCons)*

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 $p \text{ } P \equiv$
 $(\neg \text{lfinite } P \wedge (\exists a \in \text{path-inf-priorities } P.$
 $(\forall b \in \text{path-inf-priorities } P. a \leq b) \wedge \text{winning-priority } p \text{ } a))$
 $\vee (\neg \text{lnull } P \wedge \text{lfinite } P \wedge \text{llast } P \in VV \text{ } p**)$
 $\vee (\text{lnull } P \wedge p = \text{Even})$

Every path has a unique winner.

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

assumes *valid-path* P

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

proof *(cases)*

assume $\neg \text{lnull } P$


```

show ?thesis proof (cases)
  assume lfinite P
  thus ?thesis
    using assms lfinite-lset valid-path-in-V
    unfolding winning-path-def
    by auto
next
  assume ¬lfinite P
  then obtain a where a ∈ path-inf-priorities P ∧ b. b < a ⇒ b ∉ path-inf-priorities P
    using assms ex-least-nat-le[of λa. a ∈ path-inf-priorities P] path-inf-priorities-is-nonempty
    by blast
  hence ∀ q. winning-priority q a ⇔ winning-path q P
    unfolding winning-path-def using ⟨¬lnull P⟩ ⟨¬lfinite P⟩ by (metis le-antisym not-le)
  moreover have ∀ q. winning-priority p q ⇔ ¬winning-priority p** q by simp
  ultimately show ?thesis by blast
qed
qed (simp add: winning-path-def)

lemma winning-path-ltl:
  assumes P: winning-path p P ¬lnull P ¬lnull (ltl P)
  shows winning-path p (ltl P)
proof (cases)
  assume lfinite P
  moreover have llast P = llast (ltl P)
    using P(2,3) by (metis llast-LCons2 ltl-simps(2) not-lnull-conv)
  ultimately show ?thesis using P by (simp add: winning-path-def)
next
  assume ¬lfinite P
  thus ?thesis using winning-path-def path-inf-priorities-ltl P(1,2) by auto
qed

corollary winning-path-drop:
  assumes winning-path p P enat n < llength P
  shows winning-path p (ldropn n P)
using assms proof (induct n)
  case (Suc n)
  hence winning-path p (ldropn n P) using dual-order.strict-trans enat-ord-simps(2) by blast
  moreover have ltl (ldropn n P) = ldropn (Suc n) P by (simp add: ldrop-eSuc-ltl ltl-ldropn)
  moreover hence ¬lnull (ldropn n P) using Suc.premis(2) by (metis leD lnull-ldropn lnull-ltlI)
  ultimately show ?case using winning-path-ltl[of p ldropn n P] Suc.premis(2) by auto
qed simp

corollary winning-path-drop-add:
  assumes valid-path P winning-path p (ldropn n P) enat n < llength P
  shows winning-path p P
  using assms paths-are-winning-for-one-player valid-path-drop winning-path-drop by blast

lemma winning-path-LCons:
  assumes P: winning-path p P ¬lnull P
  shows winning-path p (LCons v P)
proof (cases)
  assume lfinite P

```

```

moreover have  $llast\ P = llast\ (LCons\ v\ P)$ 
using  $P(2)$  by  $(metis\ llast-LCons2\ not-lnull-conv)$ 
ultimately show  $?thesis$  using  $P$  unfolding  $winning-path-def$  by  $simp$ 
next
assume  $\neg lfinite\ P$ 
thus  $?thesis$  using  $P$   $path-inf-priorities-LCons$  unfolding  $winning-path-def$  by  $simp$ 
qed

lemma  $winning-path-supergame$ :
assumes  $winning-path\ p\ P$ 
and  $G'$ :  $ParityGame\ G'\ VV\ p^{**} \subseteq ParityGame.VV\ G'\ p^{**}\ \omega = \omega_{G'}$ 
shows  $ParityGame.winning-path\ G'\ p\ P$ 
proof—
interpret  $G'$ :  $ParityGame\ G'$  using  $G'(1)$  .
have  $\llbracket lfinite\ P; \neg lnull\ P \rrbracket \implies llast\ P \in G'.VV\ p^{**}$  and  $lnull\ P \implies p = Even$ 
using  $assms(1)$  unfolding  $winning-path-def$  using  $G'(2)$  by  $auto$ 
thus  $?thesis$  unfolding  $G'.winning-path-def$ 
using  $lnull-imp-lfinite\ assms(1)$ 
unfolding  $winning-path-def\ path-inf-priorities-def\ G'.path-inf-priorities-def\ G'(3)$ 
by  $blast$ 
qed

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  $P-not-null\ [simp]: \neg lnull\ P$ 
and  $P-valid\ [simp]: valid-path\ P$ 
and  $P-maximal\ [simp]: maximal-path\ P$ 
and  $P-v0\ [simp]: lhd\ P = v0$ 
begin
lemma  $P-LCons$ :  $P = LCons\ v0\ (ltl\ P)$  using  $lhd-LCons-ltl[OF\ P-not-null]$  by  $simp$ 

lemma  $P-len\ [simp]: enat\ 0 < llength\ P$  by  $(simp\ add: lnull-0-llength)$ 
lemma  $P-0\ [simp]: P\ \$\ 0 = v0$  by  $(simp\ add: lnth-0-conv-lhd)$ 
lemma  $P-lnth-Suc$ :  $P\ \$\ Suc\ n = ltl\ P\ \$\ n$  by  $(simp\ add: lnth-ltl)$ 
lemma  $P-no-deadends$ :  $enat\ (Suc\ n) < llength\ P \implies \neg deadend\ (P\ \$\ n)$ 
using  $valid-path-no-deadends$  by  $simp$ 
lemma  $P-no-deadend-v0$ :  $\neg lnull\ (ltl\ P) \implies \neg deadend\ v0$ 
by  $(metis\ P-LCons\ P-valid\ edges-are-in-V(2)\ not-lnull-conv\ valid-path-edges')$ 
lemma  $P-no-deadend-v0-llength$ :  $enat\ (Suc\ n) < llength\ P \implies \neg deadend\ v0$ 
by  $(metis\ P-0\ P-len\ P-valid\ enat-ord-simps(2)\ not-less-eq\ valid-path-ends-on-deadend\ zero-less-Suc)$ 
lemma  $P-ends-on-deadend$ :  $\llbracket enat\ n < llength\ P; deadend\ (P\ \$\ n) \rrbracket \implies enat\ (Suc\ n) = llength\ P$ 
using  $P-valid\ valid-path-ends-on-deadend$  by  $blast$ 

lemma  $P-lnull-ltl-deadend-v0$ :  $lnull\ (ltl\ P) \implies deadend\ v0$ 
using  $P-LCons\ maximal-no-deadend$  by  $force$ 
lemma  $P-lnull-ltl-LCons$ :  $lnull\ (ltl\ P) \implies P = LCons\ v0\ LNil$ 

```

```

    using P-LCons lnull-def by metis
lemma P-deadend-v0-LCons: deadend v0  $\implies$  P = LCons v0 LNil
    using P-lnull-ltl-LCons P-no-deadend-v0 by blast

lemma Ptl-valid [simp]: valid-path (ltl P) using valid-path-ltl by auto
lemma Ptl-maximal [simp]: maximal-path (ltl P) using maximal-ltl by auto

lemma Pdrop-valid [simp]: valid-path (ldropn n P) using valid-path-drop by auto
lemma Pdrop-maximal [simp]: maximal-path (ldropn n P) using maximal-drop by auto

lemma prefix-valid [simp]: valid-path (ltake n P)
    using valid-path-prefix[of P] by auto

lemma extension-valid [simp]: v  $\rightarrow$  v0  $\implies$  valid-path (LCons v P)
    using P-not-null P-v0 P-valid valid-path-cons by blast
lemma extension-maximal [simp]: maximal-path (LCons v P)
    by (simp add: maximal-path-cons)
lemma lappend-maximal [simp]: maximal-path (lappend P' P)
    by (simp add: maximal-path-lappend)

lemma v0-V [simp]: v0  $\in$  V by (metis P-LCons P-valid valid-path-cons-simp)
lemma v0-lset-P [simp]: v0  $\in$  lset P using P-not-null P-v0 llist.set-sel(1) by blast
lemma v0-VV: v0  $\in$  VV p  $\vee$  v0  $\in$  VV p** by simp
lemma lset-P-V [simp]: lset P  $\subseteq$  V by (simp add: valid-path-in-V)
lemma lset-ltl-P-V [simp]: lset (ltl P)  $\subseteq$  V by (simp add: valid-path-in-V)

lemma finite-llast-deadend [simp]: lfinite P  $\implies$  deadend (llast P)
    using P-maximal P-not-null maximal-ends-on-deadend by blast
lemma finite-llast-V [simp]: lfinite P  $\implies$  llast P  $\in$  V
    using P-not-null lfinite-lset lset-P-V by blast

```

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

```

lemma visits-deadend:
  assumes lset P  $\cap$  deadends p  $\neq$  {}
  shows winning-path p** P
proof-
  obtain n where n: enat n < llength P P $ n  $\in$  deadends p
  using assms by (meson lset-intersect-lnth)
  hence *: enat (Suc n) = llength P using P-ends-on-deadend unfolding deadends-def by blast
  hence llast P = P $ n by (simp add: eSuc-enat llast-conv-lnth)
  hence llast P  $\in$  deadends p using n(2) by simp
  moreover have lfinite P using * llength-eq-enat-lfiniteD by force
  ultimately show ?thesis unfolding winning-path-def deadends-def by auto
qed

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]:
 $(\bigwedge v. \llbracket v \in VV\ p; \neg \text{deadend } v \rrbracket \implies v \rightarrow \sigma\ v) \implies \text{strategy } p\ \sigma$
unfolding strategy-def **by** blast

4.2 Strategy-Conforming Paths

If *path-conforms-with-strategy* $p\ P\ \sigma$ 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 \Rightarrow 'a Path \Rightarrow 'a Strategy \Rightarrow bool **where**
 path-conforms-LNil: path-conforms-with-strategy $p\ LNil\ \sigma$
 | path-conforms-LCons-LNil: path-conforms-with-strategy $p\ (LCons\ v\ LNil)\ \sigma$
 | 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 $p\ \sigma$ **assumes** $P\text{-conforms}$ [simp]: path-conforms-with-strategy $p\ P\ \sigma$

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

locale vmc2-path = comp?: vmc-path $G\ P\ v0\ p**\ \sigma' + \text{vmc-path } G\ P\ v0\ p\ \sigma$
for $G\ P\ v0\ p\ \sigma\ \sigma'$

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 **proof**
 fix v assume $\neg \text{deadend } v$
 thus $v \rightarrow \sigma$ -arbitrary v **unfolding** σ -arbitrary-def **using** someI-ex[of $\lambda w. v \rightarrow w$] **by** blast
qed

4.4 Valid Strategies

lemma *valid-strategy-updates*: $\llbracket \text{strategy } p \ \sigma; v0 \rightarrow w0 \rrbracket \implies \text{strategy } p \ (\sigma(v0 := w0))$
unfolding strategy-def **by** auto

lemma *valid-strategy-updates-set*:
 assumes strategy $p \ \sigma \wedge v. \llbracket v \in A; v \in VV \ p; \neg \text{deadend } v \rrbracket \implies v \rightarrow \sigma' \ v$
 shows strategy p (override-on $\sigma \ \sigma' \ A$)
unfolding strategy-def **by** (metis assms override-on-def strategy-def)

lemma *valid-strategy-updates-set-strong*:
 assumes strategy $p \ \sigma$ strategy $p \ \sigma'$
 shows strategy p (override-on $\sigma \ \sigma' \ A$)
using assms(1) assms(2)[unfolded strategy-def] *valid-strategy-updates-set* **by** simp

lemma *subgame-strategy-stays-in-subgame*:
 assumes $\sigma: \text{ParityGame.strategy (subgame } V') \ p \ \sigma$
 and $v \in \text{ParityGame.VV (subgame } V') \ p \neg \text{Digraph.deadend (subgame } V') \ v$
 shows $\sigma \ v \in V'$
proof–
 interpret $G': \text{ParityGame subgame } V'$ **using** subgame-ParityGame .
 have $\sigma \ v \in V_{\text{subgame } V'}$ **using** assms **unfolding** $G'.\text{strategy-def } G'.\text{edges-are-in-}V(2)$ **by** blast
 thus $\sigma \ v \in V'$ **by** (metis Diff-iff IntE subgame-VV Player.distinct(2))
qed

lemma *valid-strategy-supergame*:
 assumes $\sigma: \text{strategy } p \ \sigma$
 and $\sigma': \text{ParityGame.strategy (subgame } V') \ p \ \sigma'$
 and $G'\text{-no-deadends: } \bigwedge v. v \in V' \implies \neg \text{Digraph.deadend (subgame } V') \ v$
 shows strategy p (override-on $\sigma \ \sigma' \ V'$) (is strategy $p \ ?\sigma$)
proof
 interpret $G': \text{ParityGame subgame } V'$ **using** subgame-ParityGame .
 fix v assume $v: v \in VV \ p \neg \text{deadend } v$
 show $v \rightarrow ?\sigma \ v$ **proof** (cases)
 assume $v \in V'$
 hence $v \in G'.VV \ p$ **using** subgame-VV $\langle v \in VV \ p \rangle$ **by** blast
 moreover have $\neg G'.\text{deadend } v$ **using** $G'\text{-no-deadends } \langle v \in V' \rangle$ **by** blast
 ultimately have $v \rightarrow_{\text{subgame } V'} \sigma' \ v$ **using** σ' **unfolding** $G'.\text{strategy-def}$ **by** blast
 moreover have $\sigma' \ v = ?\sigma \ v$ **using** $\langle v \in V' \rangle$ **by** simp
 ultimately show $?thesis$ **by** (metis subgame-E subsetCE)
 next
 assume $v \notin V'$

thus ?thesis using v σ unfolding strategy-def by simp
 qed
 qed

lemma valid-strategy-in-V: $\llbracket \text{strategy } p \ \sigma; v \in VV \ p; \neg \text{deadend } v \rrbracket \implies \sigma \ v \in V$
 unfolding strategy-def using valid-edge-set by auto

lemma valid-strategy-only-in-V: $\llbracket \text{strategy } p \ \sigma; \bigwedge v. v \in V \implies \sigma \ v = \sigma' \ v \rrbracket \implies \text{strategy } p \ \sigma'$
 unfolding strategy-def using edges-are-in-V(1) by auto

4.5 Conforming Strategies

lemma path-conforms-with-strategy-ltl [intro]:
 path-conforms-with-strategy p P $\sigma \implies \text{path-conforms-with-strategy } p \ (\text{ltl } P) \ \sigma$
 by (drule path-conforms-with-strategy.cases) (simp-all add: path-conforms-with-strategy.intros(1))

lemma path-conforms-with-strategy-drop:
 path-conforms-with-strategy p P $\sigma \implies \text{path-conforms-with-strategy } p \ (\text{ldropn } n \ P) \ \sigma$
 by (simp add: path-conforms-with-strategy-ltl ltl-ldrop[of $\lambda P. \text{path-conforms-with-strategy } p \ P \ \sigma$])

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 (coinduction arbitrary: P P')
 case (path-conforms-with-strategy P P')
 thus ?case proof (cases rule: path-conforms-with-strategy.cases)
 case path-conforms-LNil
 thus ?thesis using path-conforms-with-strategy(2) by auto
 next
 case path-conforms-LCons-LNil
 thus ?thesis by (metis lprefix-LCons-conv lprefix-antisym lprefix-code(1) path-conforms-with-strategy(2))
 next
 case (path-conforms-VVp v w)
 thus ?thesis proof (cases)
 assume $P' \neq \text{LNil} \wedge P' \neq \text{LCons } v \ \text{LNil}$
 hence $\exists Q. P' = \text{LCons } v \ (\text{LCons } w \ Q)$
 by (metis local.path-conforms-VVp(1) lprefix-LCons-conv path-conforms-with-strategy(2))
 thus ?thesis using local.path-conforms-VVp(1,3,4) path-conforms-with-strategy(2) by force
 qed auto
 next
 case (path-conforms-VVpstar v)
 thus ?thesis proof (cases)
 assume $P' \neq \text{LNil}$
 hence $\exists Q. P' = \text{LCons } v \ Q$
 using local.path-conforms-VVpstar(1) lprefix-LCons-conv path-conforms-with-strategy(2) by
 fastforce
 thus ?thesis using local.path-conforms-VVpstar path-conforms-with-strategy(2) by auto
 qed simp
 qed
 qed

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 ( $\sigma(v := w)$ )
using assms apply (coinduction arbitrary: P) by (drule path-conforms-with-strategy.cases) auto

lemma path-conforms-with-strategy-irrelevant-deadend:
  assumes path-conforms-with-strategy p P  $\sigma$  deadend v  $\vee$  v  $\notin$  VV p valid-path P
  shows path-conforms-with-strategy p P ( $\sigma(v := w)$ )
using assms proof (coinduction arbitrary: P)
  let ? $\sigma$  =  $\sigma(v := w)$ 
  case (path-conforms-with-strategy P)
  thus ?case proof (cases rule: path-conforms-with-strategy.cases)
    case (path-conforms-VVp v' w Ps)
    have w = ? $\sigma$  v' proof -
      from  $\langle$ valid-path P $\rangle$  have  $\neg$ deadend v'
      using local.path-conforms-VVp(1) valid-path-cons-simp by blast
      with assms(2) have v'  $\neq$  v using local.path-conforms-VVp(2) by blast
      thus w = ? $\sigma$  v' by (simp add: local.path-conforms-VVp(3))
    qed
  moreover
    have  $\exists P. LCons w Ps = P \wedge$  path-conforms-with-strategy p P  $\sigma \wedge$  (deadend v  $\vee$  v  $\notin$  VV p)
   $\wedge$  valid-path P
  proof -
    have valid-path (LCons w Ps)
      using local.path-conforms-VVp(1) path-conforms-with-strategy(3) valid-path-ltl' by blast
    thus ?thesis using local.path-conforms-VVp(4) path-conforms-with-strategy(2) by blast
  qed
  ultimately show ?thesis using local.path-conforms-VVp(1,2) by blast
next
  case (path-conforms-VVpstar v' Ps)
  have  $\exists P. \text{path-conforms-with-strategy } p Ps \sigma \wedge$  (deadend v  $\vee$  v  $\notin$  VV p)  $\wedge$  valid-path Ps
  using local.path-conforms-VVpstar(1,3) path-conforms-with-strategy(2,3) valid-path-ltl' by
blast
  thus ?thesis by (simp add: local.path-conforms-VVpstar(1,2))
qed simp-all
qed

lemma path-conforms-with-strategy-irrelevant-updates:
  assumes path-conforms-with-strategy p P  $\sigma \bigwedge v. v \in lset P \implies \sigma v = \sigma' v$ 
  shows path-conforms-with-strategy p P  $\sigma'$ 
using assms proof (coinduction arbitrary: P)
  case (path-conforms-with-strategy P)
  thus ?case proof (cases rule: path-conforms-with-strategy.cases)
    case (path-conforms-VVp v' w Ps)
    have w =  $\sigma' v'$  using local.path-conforms-VVp(1,3) path-conforms-with-strategy(2) by auto
    thus ?thesis using local.path-conforms-VVp(1,4) path-conforms-with-strategy(2) by auto
  qed simp-all
qed

lemma path-conforms-with-strategy-irrelevant':
  assumes path-conforms-with-strategy p P ( $\sigma(v := w)$ ) v  $\notin$  lset P
  shows path-conforms-with-strategy p P  $\sigma$ 
  by (metis assms fun-upd-triv fun-upd-upd path-conforms-with-strategy-irrelevant)

```

lemma *path-conforms-with-strategy-irrelevant-deadend'*:
assumes *path-conforms-with-strategy* p P $(\sigma(v := w))$ *deadend* $v \vee v \notin VV\ p$ *valid-path* P
shows *path-conforms-with-strategy* p P σ
by (*metis* *assms* *fun-upd-triv* *fun-upd-upd* *path-conforms-with-strategy-irrelevant-deadend*)

lemma *path-conforms-with-strategy-start*:
path-conforms-with-strategy p ($LCons\ v\ (LCons\ w\ P)$) $\sigma \implies v \in VV\ p \implies \sigma\ v = w$
by (*drule* *path-conforms-with-strategy.cases*) *simp-all*

lemma *path-conforms-with-strategy-lappend*:
assumes
 P : *lfinite* P $\neg lnull\ P$ *path-conforms-with-strategy* p P σ
and P' : $\neg lnull\ P'$ *path-conforms-with-strategy* p P' σ
and *conforms*: *llast* $P \in VV\ p \implies \sigma\ (llast\ P) = lhd\ P'$
shows *path-conforms-with-strategy* p (*lappend* $P\ P'$) σ
using *assms* **proof** (*induct* P *rule*: *lfinite-induct*)
case ($LCons\ P$)
show ?*case* **proof** (*cases*)
assume *lnull* (*ltl* P)
then obtain $v0$ **where** $v0$: $P = LCons\ v0\ LNil$
by (*metis* *LCons.premis*(1) *lhd-LCons-ltl* *llist.collapse*(1))
have *path-conforms-with-strategy* p ($LCons\ (lhd\ P)\ P'$) σ **proof** (*cases*)
assume $lhd\ P \in VV\ p$
moreover with $v0$ **have** $lhd\ P' = \sigma\ (lhd\ P)$
using *LCons.premis*(5) **by** *auto*
ultimately show ?*thesis*
using *path-conforms-VVp*[*of* $lhd\ P\ p\ lhd\ P'\ \sigma$]
by (*metis* (*no-types*) *LCons.premis*(4) $\langle \neg lnull\ P' \rangle$ *lhd-LCons-ltl*)
next
assume $lhd\ P \notin VV\ p$
thus ?*thesis* **using** *path-conforms-VVpstar* **using** *LCons.premis*(4) $v0$ **by** *blast*
qed
thus ?*thesis* **by** (*simp* *add*: $v0$)
next
assume $\neg lnull\ (ltl\ P)$
hence *: *path-conforms-with-strategy* p (*lappend* (*ltl* P) P') σ
by (*metis* *LCons.hyps*(3) *LCons.premis*(1) *LCons.premis*(2) *LCons.premis*(5) *LCons.premis*(5)
assms(4) *assms*(5) *lhd-LCons-ltl* *llast-LCons2* *path-conforms-with-strategy-ltl*)
have *path-conforms-with-strategy* p ($LCons\ (lhd\ P)\ (lappend\ (ltl\ P)\ P')$) σ **proof** (*cases*)
assume $lhd\ P \in VV\ p$
moreover hence $lhd\ (ltl\ P) = \sigma\ (lhd\ P)$
by (*metis* *LCons.premis*(1) *LCons.premis*(2) $\langle \neg lnull\ (ltl\ P) \rangle$
lhd-LCons-ltl *path-conforms-with-strategy-start*)
ultimately show ?*thesis*
using *path-conforms-VVp*[*of* $lhd\ P\ p\ lhd\ (ltl\ P)\ \sigma$] * $\langle \neg lnull\ (ltl\ P) \rangle$
by (*metis* *lappend-code*(2) *lhd-LCons-ltl*)
next
assume $lhd\ P \notin VV\ p$
thus ?*thesis* **by** (*simp* *add*: * *path-conforms-VVpstar*)
qed
with $\langle \neg lnull\ P \rangle$ **show** *path-conforms-with-strategy* p (*lappend* $P\ P'$) σ
by (*metis* *lappend-code*(2) *lhd-LCons-ltl*)


```

qed
qed simp

lemma path-conforms-with-strategy-VVpstar:
  assumes lset P  $\subseteq$  VV p**
  shows path-conforms-with-strategy p P  $\sigma$ 
using assms proof (coinduction arbitrary: P)
  case (path-conforms-with-strategy P)
  moreover have  $\bigwedge v Ps. P = LCons v Ps \implies ?case$  using path-conforms-with-strategy by auto
  ultimately show ?case by (cases P = LNil, simp) (metis lnull-def not-llnull-conv)
qed

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  $\sigma$ 
proof–
  have lset P  $\subseteq$  Vsubgame V' unfolding subgame-def using P(2) V' by auto
  with P(1) show ?thesis
  by (coinduction arbitrary: P rule: ParityGame.path-conforms-with-strategy.coinduct[OF subgame-ParityGame])
  (cases rule: path-conforms-with-strategy.cases, auto)
qed

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  $\sigma$ 
proof–
  interpret G': ParityGame subgame V' using subgame-ParityGame by blast
  show ?thesis proof
    show G'.valid-path P using subgame-valid-path P-valid P by blast
    show G'.maximal-path P using subgame-maximal-path V' P-maximal P by blast
    show G'.path-conforms-with-strategy p P  $\sigma$ 
    using subgame-path-conforms-with-strategy V' P-conforms P by blast
  qed simp-all
qed

```

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  $p\ \sigma\ \sigma'\ v0 \neq LNil$ 
  using greedy-conforming-path.disc-iff l1st.discI(1) by blast

private lemma greedy-path-lhd: greedy-conforming-path  $p\ \sigma\ \sigma'\ v0 = LCons\ v\ P \implies v = v0$ 
  using greedy-conforming-path.code by auto

private lemma greedy-path-deadend-v0: greedy-conforming-path  $p\ \sigma\ \sigma'\ v0 = LCons\ v\ P \implies P = LNil \longleftrightarrow deadend\ v0$ 
  by (metis (no-types, lifting) greedy-conforming-path.disc-iff
    greedy-conforming-path.simps(3) l1st.disc(1) l1l-simps(2))

private corollary greedy-path-deadend-v:
  greedy-conforming-path  $p\ \sigma\ \sigma'\ v0 = LCons\ v\ P \implies P = LNil \longleftrightarrow deadend\ v$ 
  using greedy-path-deadend-v0 greedy-path-lhd by metis

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

private lemma greedy-path-ltl:
  assumes greedy-conforming-path  $p\ \sigma\ \sigma'\ v0 = LCons\ v\ P$ 
  shows  $P = LNil \vee P = \text{greedy-conforming-path } p\ \sigma\ \sigma'\ (\sigma\ v0) \vee P = \text{greedy-conforming-path } p\ \sigma\ \sigma'\ (\sigma'\ v0)$ 
  apply (insert assms, frule greedy-path-lhd)
  apply (cases deadend v0, simp add: greedy-conforming-path.code)
  by (metis (no-types, lifting) greedy-conforming-path.sel(2) l1l-simps(2))

private lemma greedy-path-ltl-ex:
  assumes greedy-conforming-path  $p\ \sigma\ \sigma'\ v0 = LCons\ v\ P$ 
  shows  $P = LNil \vee (\exists v. P = \text{greedy-conforming-path } p\ \sigma\ \sigma'\ v)$ 
  using assms greedy-path-ltl by blast

private lemma greedy-path-ltl-VVp:
  assumes greedy-conforming-path  $p\ \sigma\ \sigma'\ v0 = LCons\ v0\ P\ v0 \in VV\ p\ \neg deadend\ v0$ 
  shows  $\sigma\ v0 = lhd\ P$ 
  using assms greedy-conforming-path.code by auto

private lemma greedy-path-ltl-VVpstar:
  assumes greedy-conforming-path  $p\ \sigma\ \sigma'\ v0 = LCons\ v0\ P\ v0 \in VV\ p^{**}\ \neg deadend\ v0$ 
  shows  $\sigma'\ v0 = lhd\ P$ 
  using assms greedy-conforming-path.code by auto

private lemma greedy-conforming-path-properties:
  assumes  $v0 \in V\ strategy\ p\ \sigma\ strategy\ p^{**}\ \sigma'$ 
  shows
    greedy-path-not-null:  $\neg lnull\ (\text{greedy-conforming-path } p\ \sigma\ \sigma'\ v0)$ 
    and greedy-path-v0:  $\text{greedy-conforming-path } p\ \sigma\ \sigma'\ v0\ \$\ 0 = v0$ 
    and greedy-path-valid:  $valid\text{-path } (\text{greedy-conforming-path } p\ \sigma\ \sigma'\ v0)$ 
    and greedy-path-maximal:  $maximal\text{-path } (\text{greedy-conforming-path } p\ \sigma\ \sigma'\ v0)$ 

```

and *greedy-path-conforms*: *path-conforms-with-strategy* *p* (*greedy-conforming-path* *p* σ σ' *v0*) σ
and *greedy-path-conforms'*: *path-conforms-with-strategy* *p*** (*greedy-conforming-path* *p* σ σ' *v0*) σ'
proof–
define *P* **where** [*simp*]: *P* = *greedy-conforming-path* *p* σ σ' *v0*
show $\neg \text{lnull } P \ P \ \$ \ 0 = v0$ **by** (*simp-all add: lnth-0-conv-lhd*)
{
 fix *v0* **assume** *v0* $\in V$
 let *?P* = *greedy-conforming-path* *p* σ σ' *v0*
 assume *asm*: $\neg(\exists v. ?P = LCons \ v \ LNil)$
 obtain *P'* **where** *P'*: *?P* = *LCons* *v0* *P'* **by** (*metis greedy-path-LNil greedy-path-lhd neq-LNil-conv*)
 hence $\neg \text{deadend } v0$ **using** *asm* *greedy-path-deadend-v0* $\langle v0 \in V \rangle$ **by** *blast*
 from *P'* **have** *1*: $\neg \text{lnull } P'$ **using** *asm* *l1st.collapse(1)* $\langle v0 \in V \rangle$ *greedy-path-deadend-v0* **by**
blast
 moreover from *P'* $\langle \neg \text{deadend } v0 \rangle$ *assms(2,3)* $\langle v0 \in V \rangle$
 have *v0* $\rightarrow \text{lhd } P'$
 unfolding *strategy-def* **using** *greedy-path-ltl-VVp* *greedy-path-ltl-VVpstar*
 by (*cases v0* $\in VV \ p$) *auto*
 moreover hence *lhd* *P'* $\in V$ **by** *blast*
 moreover hence $\exists v. P' = \text{greedy-conforming-path } p \ \sigma \ \sigma' \ v \wedge v \in V$
 by (*metis P' calculation(1) greedy-conforming-path.simps(2) greedy-path-ltl-ex lnull-def*)
}

The conjunction of all the above.

ultimately
 have $\exists P'. ?P = LCons \ v0 \ P' \wedge \neg \text{lnull } P' \wedge v0 \rightarrow \text{lhd } P' \wedge \text{lhd } P' \in V$
 $\wedge (\exists v. P' = \text{greedy-conforming-path } p \ \sigma \ \sigma' \ v \wedge v \in V)$
 using *P'* **by** *blast*
} **note** *coinduction-helper* = *this*

show *valid-path* *P* **using** *assms* **unfolding** *P-def*
proof (*coinduction arbitrary: v0* *rule: valid-path.coinduct*)
 case (*valid-path* *v0*)
 from $\langle v0 \in V \rangle$ *assms(2,3)* **show** *?case*
 using *coinduction-helper*[*of v0*] *greedy-path-lhd* **by** *blast*
qed

show *maximal-path* *P* **using** *assms* **unfolding** *P-def*
proof (*coinduction arbitrary: v0*)
 case (*maximal-path* *v0*)
 from $\langle v0 \in V \rangle$ *assms(2,3)* **show** *?case*
 using *coinduction-helper*[*of v0*] *greedy-path-deadend-v'* **by** *blast*
qed

{
 fix *p''* *σ''* **assume** *p''*: (*p''* = *p* \wedge *σ''* = *σ*) \vee (*p''* = *p*** \wedge *σ''* = *σ'*)
 moreover with *assms* **have** *strategy* *p''* *σ''* **by** *blast*
 hence *path-conforms-with-strategy* *p''* *P* *σ''* **using** $\langle v0 \in V \rangle$ **unfolding** *P-def*
proof (*coinduction arbitrary: v0*)
 case (*path-conforms-with-strategy* *v0*)
 show *?case* **proof** (*cases v0* $\in VV \ p''$)
}

```

case True
{ assume  $\neg(\exists v. \text{greedy-conforming-path } p \ \sigma \ \sigma' \ v0 = \text{LCons } v \ \text{LNil})$ 
  with  $\langle v0 \in V \rangle$  obtain  $P'$  where
     $P': \text{greedy-conforming-path } p \ \sigma \ \sigma' \ v0 = \text{LCons } v0 \ P' \neg \text{lnull } P' \ v0 \rightarrow \text{lhd } P'$ 
     $\text{lhd } P' \in V \ \exists v. P' = \text{greedy-conforming-path } p \ \sigma \ \sigma' \ v \wedge v \in V$ 
    using coinduction-helper by blast
  with  $\langle v0 \in VV \ p'' \rangle \ p''$  have  $\sigma'' \ v0 = \text{lhd } P'$ 
    using greedy-path-ltl-VVp greedy-path-ltl-VVpstar by blast
  with  $\langle v0 \in VV \ p'' \rangle \ P'(1,2,5)$  have ?path-conforms-VVp
    using greedy-conforming-path.code path-conforms-with-strategy(1) by fastforce
  }
thus ?thesis by auto
next
case False
thus ?thesis using coinduction-helper[of v0] path-conforms-with-strategy by auto
qed
qed
}
thus path-conforms-with-strategy  $p \ P \ \sigma \ \text{path-conforms-with-strategy } p^{**} \ P \ \sigma'$  by blast+
qed

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
  show vmc2-path  $G \ (\text{greedy-conforming-path } p \ \sigma \ \sigma' \ v0) \ v0 \ p \ \sigma \ \sigma'$ 
  using assms by unfold-locales (simp-all add: greedy-conforming-path-properties)
qed

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
  show vmc-path  $G \ (\text{greedy-conforming-path } p \ \sigma \ \sigma\text{-arbitrary } v0) \ v0 \ p \ \sigma$ 
  using assms by unfold-locales (simp-all add: greedy-conforming-path-properties)
qed

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  $p \ (\text{ltl } P) \ \sigma$ 
  using P-conforms path-conforms-with-strategy-ltl by blast
lemma Pdrop-conforms [simp]: path-conforms-with-strategy  $p \ (\text{ldropn } n \ P) \ \sigma$ 
  using P-conforms path-conforms-with-strategy-drop by blast
lemma prefix-conforms [simp]: path-conforms-with-strategy  $p \ (\text{ltake } n \ P) \ \sigma$ 
  using P-conforms path-conforms-with-strategy-prefix by blast

```

lemma *extension-conforms* [simp]:
 $(v' \in VV \ p \implies \sigma \ v' = v0) \implies \text{path-conforms-with-strategy } p \ (LCons \ v' \ P) \ \sigma$
by (metis *P-LCons P-conforms path-conforms-VVp path-conforms-VVpstar*)

lemma *extension-valid-maximal-conforming*:
assumes $v' \rightarrow v0 \ v' \in VV \ p \implies \sigma \ v' = v0$
shows $\text{vmc-path } G \ (LCons \ v' \ P) \ v' \ p \ \sigma$
using *assms* **by** *unfold-locales simp-all*

lemma *vmc-path-ldropn*:
assumes $\text{enat } n < \text{llength } P$
shows $\text{vmc-path } G \ (\text{ldropn } n \ P) \ (P \ \$ \ n) \ p \ \sigma$
using *assms* **by** *unfold-locales (simp-all add: lhd-ldropn)*

lemma *conforms-to-another-strategy*:
 $\text{path-conforms-with-strategy } p \ P \ \sigma' \implies \text{vmc-path } G \ P \ v0 \ p \ \sigma'$
using *P-not-null P-valid P-maximal P-v0* **by** *unfold-locales blast+*
end

lemma (in *ParityGame*) *valid-maximal-conforming-path-0*:
assumes $\neg \text{null } P \ \text{valid-path } P \ \text{maximal-path } P \ \text{path-conforms-with-strategy } p \ P \ \sigma$
shows $\text{vmc-path } G \ P \ (P \ \$ \ 0) \ p \ \sigma$
using *assms* **by** *unfold-locales (simp-all add: lnth-0-conv-lhd)*

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]: $\neg \text{null } (\text{ltl } P)$
using *P-LCons P-maximal maximal-no-deadend v0-no-deadend* **by** *metis*
lemma *Ptl-LCons*: $\text{ltl } P = LCons \ w0 \ (\text{ltl } (\text{ltl } P))$ **unfolding** *w0-def* **by** *simp*
lemma *P-LCons'*: $P = LCons \ v0 \ (LCons \ w0 \ (\text{ltl } (\text{ltl } P)))$ **using** *P-LCons Ptl-LCons* **by** *simp*
lemma *v0-edge-w0* [simp]: $v0 \rightarrow w0$ **using** *P-valid P-LCons'* **by** (metis *valid-path-edges'*)

lemma *Ptl-0*: $\text{ltl } P \ \$ \ 0 = \text{lhd } (\text{ltl } P)$ **by** (simp add: *lhd-conv-lnth*)
lemma *P-Suc-0*: $P \ \$ \ \text{Suc } 0 = w0$ **by** (simp add: *P-lnth-Suc Ptl-0 w0-def*)
lemma *Ptl-edge* [simp]: $v0 \rightarrow \text{lhd } (\text{ltl } P)$ **by** (metis *P-LCons' P-valid valid-path-edges' w0-def*)

lemma *v0-conforms*: $v0 \in VV \ p \implies \sigma \ v0 = w0$
using *path-conforms-with-strategy-start* **by** (metis *P-LCons' P-conforms*)

lemma *w0-V* [simp]: $w0 \in V$ **by** (metis *Ptl-LCons Ptl-valid valid-path-cons-simp*)
lemma *w0-lset-P* [simp]: $w0 \in \text{lset } P$ **by** (metis *P-LCons' lset-intros(1) lset-intros(2)*)

lemma *vmc-path-ltl* [simp]: $\text{vmc-path } G \ (\text{ltl } P) \ w0 \ p \ \sigma$ **by** (unfold-locales) (simp-all add: *w0-def*)

end

context *vmc-path* **begin**

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

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

using *P-0 P-no-deadends* **by** (*unfold-locales*) (*metis enat-ltl-Suc lnull-0-llength*)

lemma *vmc-path-conforms*:

assumes *enat* (*Suc n*) < *llength P P \$ n* $\in VV \ p$

shows $\sigma (P \$ n) = P \$ \text{Suc } n$

proof–

define *P'* **where** *P' = ldropn n P*

then interpret *P'*: *vmc-path G P' P \$ n p σ* **using** *vmc-path-ldropn* *assms(1)* *Suc-llength* **by** *blast*

have $\neg \text{deadend} (P \$ n)$ **using** *assms(1)* *P-no-deadends* **by** *blast*

then interpret *P'*: *vmc-path-no-deadend G P' P \$ n p σ* **by** *unfold-locales*

have $\sigma (P \$ n) = P'.w0$ **using** *P'.v0-conforms* *assms(2)* **by** *blast*

thus *?thesis* **using** *P'-def P'.P-Suc-0* *assms(1)* **by** *simp*

qed

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

fix *v* **assume** $v \in \text{lset } P$

thus $v \in S$ **using** *vmc-path-axioms* *assms(1,2)* **proof** (*induct arbitrary: v0 rule: llist-set-induct*)

case (*find P*)

then interpret *vmc-path G P v0 p σ* **by** *blast*

show *?case* **by** (*simp add: find.premis(3)*)

next

case (*step P v*)

then interpret *vmc-path G P v0 p σ* **by** *blast*

show *?case* **proof** (*cases*)

assume $\text{lnull} (\text{ltl } P)$

hence $P = LCons \ v \ LNil$ **by** (*metis llist.disc(2) lset-cases step.hyps(2)*)

thus *?thesis* **using** *step.premis(3)* *P-LCons* **by** *blast*

next

assume $\neg \text{lnull} (\text{ltl } P)$

then interpret *vmc-path-no-deadend G P v0 p σ*

using *vmc-path-lnull-ltl-no-deadend* **by** *blast*

show $v \in S$

using *step.hyps(3)*

step-assumption[*OF vmc-path-no-deadend-axioms* $\langle v0 \in S \rangle \langle Q \ P \rangle$]

vmc-path-ltl

```

      by blast
    qed
  qed
qed

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

corollary vmc-path-lset-induction-simple [case-names base step]:
  assumes base:  $v0 \in S$ 
  and step:  $\bigwedge P \ v0. \llbracket vmc\text{-}path\text{-}no\text{-}deadend \ G \ P \ v0 \ p \ \sigma; v0 \in S \rrbracket \implies vmc\text{-}path\text{-}no\text{-}deadend.w0 \ P \in S$ 
  shows  $lset \ P \subseteq S$ 
  using assms vmc-path-lset-induction[of  $\lambda P. \ True$ ] by blast

Another induction schema for proving  $lset \ P \subseteq S$  based on closure properties.

lemma vmc-path-lset-induction-closed-subset [case-names VVp VVpstar v0 disjoint]:
  assumes VVp:  $\bigwedge v. \llbracket v \in S; \neg deadend \ v; v \in VV \ p \rrbracket \implies \sigma \ v \in S \cup T$ 
  and VVpstar:  $\bigwedge v \ w. \llbracket v \in S; \neg deadend \ v; v \in VV \ p^{**}; v \rightarrow w \rrbracket \implies w \in S \cup T$ 
  and v0:  $v0 \in S$ 
  and disjoint:  $lset \ P \cap T = \{\}$ 
  shows  $lset \ P \subseteq S$ 
using disjoint proof (induct rule: vmc-path-lset-induction)
  case (step  $P \ v0$ )
  interpret vmc-path-no-deadend  $G \ P \ v0 \ p \ \sigma$  using step.hyps(1) .
  have  $lset \ (ltl \ P) \cap T = \{\}$  using step.hyps(3)
  by (meson disjoint-eq-subset-Compl lset-ltl order.trans)
  moreover have  $w0 \in S \cup T$ 
  using assms(1,2)[of  $v0$ ] step.hyps(2) v0-no-deadend v0-conforms
  by (cases  $v0 \in VV \ p$ ) simp-all
  ultimately show ?case using step.hyps(3) w0-lset-P by blast
qed (insert v0)

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

lemma *visits-via-monotone*: $\llbracket \text{visits-via } P A W; A \subseteq A' \rrbracket \implies \text{visits-via } P A' W$
unfolding *visits-via-def* **by** *blast*

lemma *visits-via-visits*: $\text{visits-via } P A W \implies \text{lset } P \cap W \neq \{\}$
unfolding *visits-via-def* **by** (*meson disjoint-iff-not-equal in-lset-conv-lnth*)

lemma (*in vmc-path*) *visits-via-trivial*: $v0 \in W \implies \text{visits-via } P A W$
unfolding *visits-via-def* **apply** (*rule exI[of - 0]*) **using** *zero-enat-def* **by** *auto*

lemma *visits-via-LCons*:
assumes *visits-via* $P A W$
shows *visits-via* ($LCons v0 P$) ($\text{insert } v0 A$) W
proof–
obtain n **where** $n: \text{enat } n < \text{llength } P P \$ n \in W \text{ lset } (\text{ltake } (\text{enat } n) P) \subseteq A$
using *assms* **unfolding** *visits-via-def* **by** *blast*
define P' **where** $P' = LCons v0 P$
have $\text{enat } (Suc\ n) < \text{llength } P'$ **unfolding** P' -*def*
by (*metis n(1) ldropsn-Suc-LCons ldropsn-Suc-conv-ldropsn ldropsn-eq-LConsD*)
moreover **have** $P' \$ Suc\ n \in W$ **unfolding** P' -*def* **by** (*simp add: n(2)*)
moreover **have** $\text{lset } (\text{ltake } (\text{enat } (Suc\ n)) P') \subseteq \text{insert } v0 A$
using *lset-ltake-Suc[of P' v0 n A]* **unfolding** P' -*def* **by** (*simp add: n(3)*)
ultimately show *?thesis* **unfolding** *visits-via-def* P' -*def* **by** *blast*
qed

lemma (*in vmc-path-no-deadend*) *visits-via-ltl*:
assumes *visits-via* $P A W$
and $v0: v0 \notin W$
shows *visits-via* ($\text{ltl } P$) $A W$
proof–
obtain n **where** $n: \text{enat } n < \text{llength } P P \$ n \in W \text{ lset } (\text{ltake } (\text{enat } n) P) \subseteq A$
using *assms(1)[unfolding visits-via-def]* **by** *blast*
have $n \neq 0$ **using** $v0\ n(2)\ \text{DiffE}$ **by** *force*
then obtain n' **where** $n': Suc\ n' = n$ **using** *nat.exhaust* **by** *metis*
have $\exists n. \text{enat } n < \text{llength } (\text{ltl } P) \wedge (\text{ltl } P) \$ n \in W \wedge \text{lset } (\text{ltake } (\text{enat } n) (\text{ltl } P)) \subseteq A$
apply (*rule exI[of - n']*)
using $n\ n'\ \text{enat-Suc-ltl}[of\ n'\ P]\ P\text{-lnth-Suc}\ \text{lset-ltake-ltl}[of\ n'\ P]$ **by** *auto*
thus *?thesis* **using** *visits-via-def* **by** *blast*
qed

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

5.2 Attracting Strategy from a Single Node

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

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

lemma (in *vmc-path*) *strategy-attracts-viaE*:
assumes *strategy-attracts-via* $p \ \sigma \ v0 \ A \ W$
shows *visits-via* $P \ A \ W$
using *strategy-attracts-via-def* *assms* *vmc-path-axioms* **by** *blast*

lemma (in *vmc-path*) *strategy-attracts-via-SucE*:
assumes *strategy-attracts-via* $p \ \sigma \ v0 \ A \ W \ v0 \notin W$
shows $\exists n. \text{enat } (Suc \ n) < \text{llength } P \wedge P \ \$ \ Suc \ n \in W \wedge \text{lset } (\text{ltake } (\text{enat } (Suc \ n)) \ P) \subseteq A$
proof–
obtain n **where** $n: \text{enat } n < \text{llength } P \ P \ \$ \ n \in W \ \text{lset } (\text{ltake } (\text{enat } n) \ P) \subseteq A$
using *strategy-attracts-viaE* [*unfolded visits-via-def*] *assms*(1) **by** *blast*
have $n \neq 0$ **using** *assms*(2) $n(2)$ **by** (*metis* *P-0*)
thus *?thesis* **using** *n not0-implies-Suc* **by** *blast*
qed

lemma (in *vmc-path*) *strategy-attracts-via-lset*:
assumes *strategy-attracts-via* $p \ \sigma \ v0 \ A \ W$
shows $\text{lset } P \cap W \neq \{\}$
using *assms* [*THEN strategy-attracts-viaE*, *unfolded visits-via-def*]
by (*meson disjoint-iff-not-equal lset-lnth-member subset-refl*)

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 $v0 \in A \cup W$
proof–
obtain P **where** *vmc-path* $G \ P \ v0 \ p \ \sigma$ **using** *strategy-conforming-path-exists-single* *assms* **by** *blast*
then interpret *vmc-path* $G \ P \ v0 \ p \ \sigma$.
obtain n **where** $n: \text{enat } n < \text{llength } P \ P \ \$ \ n \in W \ \text{lset } (\text{ltake } (\text{enat } n) \ P) \subseteq A$
using $\sigma(2)$ [*unfolded strategy-attracts-via-def visits-via-def*] *vmc-path-axioms* **by** *blast*
show *?thesis* **proof** (*cases* $n = 0$)
case *True* **thus** *?thesis* **using** $n(2)$ **by** *simp*
next
case *False*
hence $\text{lhd } (\text{ltake } (\text{enat } n) \ P) = \text{lhd } P$ **by** (*simp add: enat-0-iff*(1))
hence $v0 \in \text{lset } (\text{ltake } (\text{enat } n) \ P)$
by (*metis* $\langle n \neq 0 \rangle$ *P-not-null P-v0 enat-0-iff*(1) *lset.set-sel*(1) *ltake.disc*(2))
thus *?thesis* **using** $n(3)$ **by** *blast*
qed
qed

corollary *strategy-attracts-not-outside*:
 $\llbracket v0 \in V - A - W; \text{strategy } p \ \sigma \rrbracket \implies \neg \text{strategy-attracts-via } p \ \sigma \ v0 \ A \ W$
using *strategy-attracts-via-v0* **by** *blast*

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* $p \ \sigma \ v0 \ A \ W$
unfolding *strategy-attracts-via-def* **using** *assms* **by** *blast*

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
  assume  $\text{deadend } v$ 
  define  $P$  where  $[simp]: P = LCons \ v \ LNil$ 
  interpret vmc-path  $G \ P \ v \ p \ \sigma$  proof
    show valid-path  $P$  using  $\langle v \in A - W \rangle \ \langle v \in V \rangle$  valid-path-base' by auto
    show maximal-path  $P$  using  $\langle \text{deadend } v \rangle$  by (simp add: maximal-path.intros(2))
    show path-conforms-with-strategy  $p \ P \ \sigma$  by (simp add: path-conforms-LCons-LNil)
  qed simp-all
  have visits-via  $P \ A \ W$  using assms(3) strategy-attracts-viaE by blast
  moreover have length  $P = eSuc \ 0$  by simp
  ultimately have  $P \ \$ \ 0 \in W$  by (simp add: enat-0-iff(1) visits-via-def)
  with  $\langle v \in A - W \rangle$  show False by auto
qed

```

lemma *attractor-strategy-on-extends*:

```

 $\llbracket \text{strategy-attracts-via } p \ \sigma \ v0 \ A \ W; A \subseteq A' \rrbracket \implies \text{strategy-attracts-via } p \ \sigma \ v0 \ A' \ W$ 
unfolding strategy-attracts-via-def using visits-via-monotone by blast

```

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

```

proof
  fix  $P$  assume  $v0 \in W$  vmc-path  $G \ P \ v0 \ p \ \sigma$ 
  then interpret vmc-path  $G \ P \ v0 \ p \ \sigma$  by blast
  show visits-via  $P \ A \ W$  using visits-via-trivial using  $\langle v0 \in W \rangle$  by blast
qed

```

lemma *strategy-attracts-via-successor*:

```

assumes  $\sigma: \text{strategy } p \ \sigma$  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
  fix  $P$  assume vmc-path  $G \ P \ w0 \ p \ \sigma$ 
  then interpret vmc-path  $G \ P \ w0 \ p \ \sigma$  .
  define  $P'$  where  $[simp]: P' = LCons \ v0 \ P$ 
  then interpret  $P': \text{vmc-path } G \ P' \ v0 \ p \ \sigma$ 
    using extension-valid-maximal-conforming  $w0$  by blast
  interpret  $P': \text{vmc-path-no-deadend } G \ P' \ v0 \ p \ \sigma$  using  $\langle v0 \rightarrow w0 \rangle$  by unfold-locales blast
  have visits-via  $P' \ A \ W$  using  $\sigma(2)$   $P'.\text{strategy-attracts-viaE}$  by blast
  thus visits-via  $P \ A \ W$  using  $P'.\text{visits-via-ltl } v0$  by simp
qed

```

lemma *strategy-attracts-VVp*:

```

assumes  $\sigma: \text{strategy } p \ \sigma$  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–
  have  $v0 \rightarrow \sigma \ v0$  using  $\sigma(1)[\text{unfolded strategy-def}] \ v(2,3)$  by blast
  hence strategy-attracts-via  $p \ \sigma \ (\sigma \ v0) \ A \ W$ 
    using strategy-attracts-via-successor  $\sigma \ v(1)$  by blast
  thus ?thesis using strategy-attracts-via-v0  $\langle v0 \rightarrow \sigma \ v0 \rangle \ \sigma(1)$  by blast

```

qed

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 $\neg v0 \rightarrow w0$

by (*metis* *assms* *strategy-attracts-not-outside* *strategy-attracts-via-successor*)

5.3 Attracting strategy from a set of nodes

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

definition *strategy-attracts* :: *Player* \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* $P A W$

using *assms*(1)[*unfolded strategy-attracts-def*] *assms*(2) *strategy-attracts-viaE* **by** *blast*

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

assumes $\bigwedge P v. \llbracket v \in A; \text{vmc-path } G P v p \sigma \rrbracket \implies \text{visits-via } P A W$

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

unfolding *strategy-attracts-def* **using** *assms* **by** *blast*

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

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

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

using *assms*(1)[*unfolded strategy-attracts-def*] *assms*(2) *strategy-attracts-via-lset*(1)[*of A W*]

by *blast*

lemma *strategy-attracts-empty* [*simp*]: *strategy-attracts* $p \sigma \{\}$ W **by** *blast*

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** $\neg ?A$)

proof

assume $?A$

then obtain n **where** $n: \text{enat } n < \text{llength } P$ $n \in W$ *lset* (*ltake* (*enat* n) P) $\subseteq A$

unfolding *visits-via-def* **by** *blast*

have $n \neq 0$ **using** $\langle v \in A - W \rangle n(2)$ $P(1)$ *DiffD2* **by** *force*

moreover have $n \neq \text{Suc } 0$ **using** $\langle w \notin A \cup W \rangle n(2)$ $P(1)$ **by** *auto*

ultimately have $\text{Suc } (\text{Suc } 0) \leq n$ **by** *presburger*

hence *lset* (*ltake* (*enat* ($\text{Suc } (\text{Suc } 0)$)) P) $\subseteq A$ **using** $n(3)$

by (*meson* *contra-subsetD* *enat-ord-simps*(1) *lset-ltake-prefix* *lset-lnth-member* *lset-subset*)

moreover have $\text{enat } (\text{Suc } 0) < \text{llength } (\text{ltake } (e\text{Suc } (e\text{Suc } 0)) P)$ **proof**–

have $*$: $\text{enat } (\text{Suc } (\text{Suc } 0)) < \text{llength } P$

using $\langle \text{Suc } (\text{Suc } 0) \leq n \rangle n(1)$ **by** (*meson* *enat-ord-simps*(2) *le-less-linear* *less-le-trans* *neq-iff*)

have $\text{llength } (\text{ltake } (\text{enat } (\text{Suc } (\text{Suc } 0))) P) = \min (\text{enat } (\text{Suc } (\text{Suc } 0))) (\text{llength } P)$ **by** *simp*

hence $\text{llength } (\text{ltake } (\text{enat } (\text{Suc } (\text{Suc } 0))) P) = \text{enat } (\text{Suc } (\text{Suc } 0))$

using $*$ **by** (*simp* *add: min-absorb1*)

thus $?thesis$ **by** (*simp* *add: eSuc-enat zero-enat-def*)

```

qed
ultimately have ltake (enat (Suc (Suc 0))) P $ Suc 0 ∈ A by (simp add: lset-lnth-member)
hence P $ Suc 0 ∈ A by (simp add: lnth-ltake)
thus False using P(1,3) by auto
qed

```

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

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

```

assumes  $\sigma$ : strategy-attracts p  $\sigma$  A W strategy p  $\sigma$ 
and  $v$ :  $v \rightarrow w$   $v \in A - W$   $w \notin A \cup W$ 
shows  $v \in VV\ p \wedge \sigma\ v \neq w$ 
proof (rule ccontr)
assume contra:  $\neg(v \in VV\ p \wedge \sigma\ v \neq w)$ 

define  $\sigma'$  where  $\sigma' = \sigma$ -arbitrary( $v := w$ )
hence strategy p**  $\sigma'$  using  $\langle v \rightarrow w \rangle$  by (simp add: valid-strategy-updates)
then obtain P where P: vmc2-path G P v p  $\sigma\ \sigma'$ 
using  $\langle v \rightarrow w \rangle$  strategy-conforming-path-exists  $\sigma(2)$  by blast
then interpret vmc2-path G P v p  $\sigma\ \sigma'$  .
interpret vmc-path-no-deadend G P v p  $\sigma$  using  $\langle v \rightarrow w \rangle$  by unfold-locales blast
interpret comp: vmc-path-no-deadend G P v p**  $\sigma'$  using  $\langle v \rightarrow w \rangle$  by unfold-locales blast
have  $w = w0$  using contra  $\sigma'$ -def v0-conforms comp.v0-conforms by (cases  $v \in VV\ p$ ) auto
hence  $\neg$ visits-via P A W
using strategy-attracts-invalid-path[of P v w ltl (ltl P)] v(2,3) P-LCons' by simp
thus False by (meson DiffE  $\sigma(1)$  strategy-attractsE v(2))
qed

```

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

lemma *strategy-attracts-irrelevant-override:*

```

assumes strategy-attracts p  $\sigma$  A W strategy p  $\sigma$  strategy p  $\sigma'$ 
shows strategy-attracts p (override-on  $\sigma'\ \sigma$  ( $A - W$ )) A W
proof (rule strategy-attractsI, rule ccontr)
fix P v
let  $?\sigma =$  override-on  $\sigma'\ \sigma$  ( $A - W$ )
assume vmc-path G P v p  $?\sigma$ 
then interpret vmc-path G P v p  $?\sigma$  .
assume  $v \in A$ 
hence P $ 0 ∈ A using  $\langle v \in A \rangle$  by simp
moreover assume contra:  $\neg$ visits-via P A W
ultimately have P $ 0 ∈ A - W unfolding visits-via-def by (meson DiffI P-len not-less0 lset-ltake)
have  $\neg$ lset P ⊆ A - W proof
assume lset P ⊆ A - W
hence  $\bigwedge v. v \in \text{lset } P \implies \text{override-on } \sigma'\ \sigma$  ( $A - W$ )  $v = \sigma\ v$  by auto
hence path-conforms-with-strategy p P  $\sigma$ 
using path-conforms-with-strategy-irrelevant-updates[OF P-conforms] by blast
hence vmc-path G P (P $ 0) p  $\sigma$ 
using conforms-to-another-strategy P-0 by blast

```

thus *False*
 using *contra* $\langle P \$ 0 \in A \rangle$ *assms*(1)
 by (*meson vmc-path.strategy-attractsE*)
 qed
 hence $\exists n. \text{enat } n < \text{llength } P \wedge P \$ n \notin A - W$ by (*meson lset-subset*)
 then obtain *n* where *n*: $\text{enat } n < \text{llength } P \wedge P \$ n \notin A - W$
 $\bigwedge i. i < n \implies \neg(\text{enat } i < \text{llength } P \wedge P \$ i \notin A - W)$
 using *ex-least-nat-le*[of $\lambda n. \text{enat } n < \text{llength } P \wedge P \$ n \notin A - W$] by *blast*
 hence *n-min*: $\bigwedge i. i < n \implies P \$ i \in A - W$
 using *dual-order.strict-trans enat-ord-simps*(2) by *blast*
 have $n \neq 0$ using $\langle P \$ 0 \in A - W \rangle$ *n*(1) by *meson*
 then obtain *n'* where *n'*: $\text{Suc } n' = n$ using *not0-implies-Suc* by *blast*
 hence $P \$ n' \in A - W$ using *n-min* by *blast*
 moreover have $P \$ n' \rightarrow P \$ \text{Suc } n'$ using *P-valid n*(1) *n'* *valid-path-edges* by *blast*
 moreover have $P \$ \text{Suc } n' \notin A \cup W$ **proof**–
 have $P \$ n \notin W$ using *contra n*(1) *n-min* **unfolding** *visits-via-def*
 by (*meson Diff-subset lset-ltake subsetCE*)
 thus *?thesis* using *n*(1) *n'* by *blast*
 qed
 ultimately have $P \$ n' \in VV p \wedge \sigma (P \$ n') \neq P \$ \text{Suc } n'$
 using *strategy-attracts-does-not-leave*[of $p \ \sigma \ A \ W \ P \$ n' \ P \$ \text{Suc } n'$]
assms(1,2) by *blast*
 thus *False*
 using *n*(1) *n'* *vmc-path-conforms* $\langle P \$ n' \in A - W \rangle$ by (*metis override-on-apply-in*)
 qed

lemma *strategy-attracts-trivial* [*simp*]: *strategy-attracts* $p \ \sigma \ W \ W$
 by (*simp add: strategy-attracts-def strategy-attracts-via-trivial*)

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

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

assumes $W \subseteq V$
 and σ : *strategy-attracts* $p \ \sigma \ A \ W$
 and *P-hits-A*: $\text{lset } P \cap A \neq \{\}$
 shows $\text{lset } P \cap W \neq \{\}$

proof–

obtain *n* where *n*: $\text{enat } n < \text{llength } P \wedge P \$ n \in A$ using *P-hits-A* by (*meson lset-intersect-lnth*)
 define *P'* where $P' = \text{ldropn } n \ P$
 interpret *vmc-path* $G \ P' \ P \$ n \ p \ \sigma$ **unfolding** *P'-def* using *vmc-path-ldropn n*(1) by *blast*
 have *visits-via P' A W* using $\sigma \ n$ (2) *strategy-attractsE* by *blast*
 thus *?thesis* **unfolding** *P'-def* using *visits-via-visits in-lset-ldropnD*[of $- \ n \ P$] by *blast*

qed

lemma *attracted-strategy-step*:

assumes σ : *strategy* $p \ \sigma \ \text{strategy-attracts } p \ \sigma \ A \ W$
 and *v0*: $\neg \text{deadend } v0 \wedge v0 \in A - W \wedge v0 \in VV p$
 shows $\sigma \ v0 \in A \cup W$
 by (*metis DiffD1 strategy-attracts-VVp assms strategy-attracts-def*)

lemma (in *vmc-path-no-deadend*) *attracted-path-step*:

assumes σ : *strategy-attracts* $p \ \sigma \ A \ W$
 and *v0*: $v0 \in A - W$

```

shows  $w0 \in A \cup W$ 
by (metis (no-types) DiffD1 P-LCons'  $\sigma$  strategy-attractsE strategy-attracts-invalid-path v0)

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  $p\ S \equiv \{v \in V - S. \neg \text{deadend } v \wedge$ 
     $(v \in VV\ p \longrightarrow (\exists w. v \rightarrow w \wedge w \in S))$ 
     $\wedge (v \in VV\ p^{**} \longrightarrow (\forall w. v \rightarrow w \longrightarrow w \in S))\}$ 
```

```
abbreviation attractor-step  $p\ W\ S \equiv W \cup S \cup \text{directly-attracted } p\ S$ 
```

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

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

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]:  $v \in \text{directly-attracted } p\ W \implies \neg \text{deadend } v$ 
unfolding directly-attracted-def by blast+
```

6.2 *attractor-step*

```
lemma attractor-step-empty: attractor-step  $p\ \{\} \ \{\} = \{\}$ 
and attractor-step-bounded-by-V:  $\llbracket W \subseteq V; S \subseteq V \rrbracket \implies \text{attractor-step } p\ W\ S \subseteq V$ 
by simp-all
```

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  $p\ W$ )
unfolding directly-attracted-def by (rule monoI) auto
```

6.3 Basic Properties of an Attractor

```

lemma attractor-unfolding: attractor p W = attractor-step p W (attractor p W)
  unfolding attractor-def using attractor-step-mono lfp-unfold by blast
lemma attractor-lowerbound: attractor-step p W S ⊆ S ⇒ attractor p W ⊆ S
  unfolding attractor-def using attractor-step-mono by (simp add: lfp-lowerbound)
lemma attractor-set-non-empty: W ≠ {} ⇒ attractor p W ≠ {}
  and attractor-set-base: W ⊆ attractor p W
  using attractor-unfolding by auto
lemma attractor-in-V: W ⊆ V ⇒ attractor p W ⊆ V
  using attractor-lowerbound attractor-step-bounded-by-V by auto

```

6.4 Attractor Set Extensions

```

lemma attractor-set-VVp:
  assumes v ∈ VV p v→w w ∈ attractor p W
  shows v ∈ attractor p W
  apply (subst attractor-unfolding) unfolding directly-attracted-def using assms by auto

lemma attractor-set-VVpstar:
  assumes ¬deadend v ∧ w. v→w ⇒ w ∈ attractor p W
  shows v ∈ attractor p W
  apply (subst attractor-unfolding) unfolding directly-attracted-def using assms by auto

```

6.5 Removing an Attractor

```

lemma removing-attractor-induces-no-deadends:
  assumes v ∈ S - attractor p W v→w w ∈ S ∧ w. [ v ∈ VV p** ; v→w ] ⇒ w ∈ S
  shows ∃ w ∈ S - attractor p W. v→w
proof-
  have v ∈ V using ⟨v→w⟩ by blast
  thus ?thesis proof (cases rule: VV-cases)
    assume v ∈ VV p
    thus ?thesis using attractor-set-VVp assms by blast
  next
    assume v ∈ VV p**
    thus ?thesis using attractor-set-VVpstar assms by (metis Diff-iff edges-are-in-V(2))
  qed
qed

```

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

```

lemma subgame-without-deadends:
  assumes V'-def: V' = V - attractor p (deadends p**) - attractor p** (deadends p****)
  (is V' = V - ?A - ?B)
  and v: v ∈ V_subgame V'
  shows ¬Digraph.deadend (subgame V') v
proof (cases)
  assume deadend v
  have v: v ∈ V - ?A - ?B using v unfolding V'-def subgame-def by simp
  { fix p' assume v ∈ VV p'**
    hence v ∈ attractor p' (deadends p'**)
    using ⟨deadend v⟩ attractor-set-base[of deadends p' p']
  }

```

```

    unfolding deadends-def by blast
    hence False using v by (cases p'; cases p) auto
  }
  thus ?thesis using v by blast
next
assume ¬deadend v
have v: v ∈ V - ?A - ?B using v unfolding V'-def subgame-def by simp
define G' where G' = subgame V'
interpret G': ParityGame G' unfolding G'-def using subgame-ParityGame .
show ?thesis proof
  assume Digraph.deadend (subgame V') v
  hence G'.deadend v unfolding G'-def .
  have all-in-attractor:  $\bigwedge w. v \rightarrow w \implies w \in ?A \vee w \in ?B$  proof (rule ccontr)
    fix w
    assume v → w ¬(w ∈ ?A ∨ w ∈ ?B)
    hence w ∈ V' unfolding V'-def by blast
    hence w ∈ VG' unfolding G'-def subgame-def using ⟨v → w⟩ by auto
    hence v →G' w using ⟨v → w⟩ assms(2) unfolding G'-def subgame-def by auto
    thus False using ⟨G'.deadend v⟩ using ⟨w ∈ VG'⟩ by blast
  qed
  { fix p' assume v ∈ VV p'
    { assume  $\exists w. v \rightarrow w \wedge w \in \text{attractor } p' \text{ (deadends } p'^{**})$ 
      hence v ∈ attractor p' (deadends p'**) using ⟨v ∈ VV p'⟩ attractor-set-VVp by blast
      hence False using v by (cases p'; cases p) auto
    }
    hence  $\bigwedge w. v \rightarrow w \implies w \in \text{attractor } p'^{**} \text{ (deadends } p'^{****})$ 
      using all-in-attractor by (cases p'; cases p) auto
    hence v ∈ attractor p' (deadends p'****)
      using ⟨¬deadend v⟩ ⟨v ∈ VV p'⟩ attractor-set-VVpstar by auto
    hence False using v by (cases p'; cases p) auto
  }
  thus False using v by blast
qed
qed

```

6.6 Attractor Set Induction

lemma *mono-restriction-is-mono*: $\text{mono } f \implies \text{mono } (\lambda S. f (S \cap V))$
 unfolding mono-def by (meson inf-mono monoD subset-refl)

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 $P (\text{attractor } p \ W)$
proof –
 let ?P = $\lambda S. P (S \cap V)$
 let ?f = $\lambda S. \text{attractor-step } p \ W (S \cap V)$


```

let ?A = lfp ?f
let ?B = lfp (attractor-step p W)
have f-mono: mono ?f
  using mono-restriction-is-mono[of attractor-step p W] attractor-step-mono by simp
have P-A: ?P ?A proof (rule lfp-ordinal-induct-set)
  show  $\bigwedge S. ?P S \implies ?P (W \cup (S \cap V) \cup \text{directly-attracted } p (S \cap V))$ 
    by (metis assms(1) attractor-step-bounded-by-V inf.absorb1 inf-le2 local.step)
  show  $\bigwedge M. \forall S \in M. ?P S \implies ?P (\bigcup M)$  proof –
    fix M
    let ?M = {S  $\cap$  V | S. S  $\in$  M}
    assume  $\forall S \in M. ?P S$ 
    hence  $\forall S \in ?M. S \subseteq V \wedge P S$  by auto
    hence *: P ( $\bigcup ?M$ ) by (simp add: union)
    have  $\bigcup ?M = (\bigcup M) \cap V$  by blast
    thus ?P ( $\bigcup M$ ) using * by auto
  qed
qed (insert f-mono)

have *:  $W \cup (V \cap V) \cup \text{directly-attracted } p (V \cap V) \subseteq V$ 
  using  $\langle W \subseteq V \rangle$  attractor-step-bounded-by-V by auto
have ?A  $\subseteq V$  ?B  $\subseteq V$  using * by (simp-all add: lfp-lowerbound)

have ?A = ?f ?A using f-mono lfp-unfold by blast
hence ?A =  $W \cup (?A \cap V) \cup \text{directly-attracted } p (?A \cap V)$  using  $\langle ?A \subseteq V \rangle$  by simp
hence *: attractor-step p W ?A  $\subseteq ?A$  using  $\langle ?A \subseteq V \rangle$  inf.absorb1 by fastforce

have ?B = attractor-step p W ?B using attractor-step-mono lfp-unfold by blast
hence ?f ?B  $\subseteq ?B$  using  $\langle ?B \subseteq V \rangle$  by (metis (no-types, lifting) equalityD2 le-iff-inf)

have ?A = ?B proof
  show ?A  $\subseteq ?B$  using  $\langle ?f ?B \subseteq ?B \rangle$  by (simp add: lfp-lowerbound)
  show ?B  $\subseteq ?A$  using * by (simp add: lfp-lowerbound)
qed
hence ?P ?B using P-A by (simp add: attractor-def)
thus ?thesis using  $\langle ?B \subseteq V \rangle$  by (simp add: attractor-def le-iff-inf)
qed

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 σ -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* $p \ \sigma \ v0$
unfolding *winning-strategy-def* **using** *assms* **by** *blast*

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

assumes σ : *winning-strategy* $p \ \sigma \ v$

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

shows *winning-path* $p \ P$

proof–

obtain n **where** n : $\text{enat } n < \text{length } P \ P \ \$ \ n = v$ **using** v **by** (*meson in-lset-conv-lnth*)

interpret P' : *vmc-path* $G \ \text{ldropn } n \ P \ v \ p \ \sigma$ **using** n *vmc-path-ldropn* **by** *blast*

have *winning-path* $p \ (\text{ldropn } n \ P)$ **using** σ **by** (*simp add: winning-strategy-def P'.vmc-path-axioms*)

thus *?thesis* **using** *winning-path-drop-add P-valid n(1)* **by** *blast*

qed

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

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

assumes σ : *strategy* $p \ \sigma$ *winning-strategy* $p \ \sigma \ v$

and σ' : *strategy* $p \ \sigma'$ *winning-strategy* $p \ \sigma' \ v$

and v : $v \in V$

shows *False*

proof–

obtain P **where** *vmc2-path* $G \ P \ v \ p \ \sigma \ \sigma'$ **using** *assms strategy-conforming-path-exists* **by** *blast*

then interpret *vmc2-path* $G \ P \ v \ p \ \sigma \ \sigma'$.

have *winning-path* $p \ P$

using *paths-hits-winning-strategy-is-winning* $\sigma(2) \ v0\text{-lset-}P$ **by** *blast*

moreover have *winning-path* $p \ P$

using *comp.paths-hits-winning-strategy-is-winning* $\sigma'(2) \ v0\text{-lset-}P$ **by** *blast*

ultimately show *False* **using** $P\text{-valid paths-are-winning-for-one-player}$ **by** *blast*

qed

7.1 Deadends

lemma *no-winning-strategy-on-deadends*:

assumes $v \in VV \ p \ \text{deadend } v \ \text{strategy } p \ \sigma$

shows $\neg \text{winning-strategy } p \ \sigma \ v$

proof–

obtain P **where** *vmc-path* $G \ P \ v \ p \ \sigma$ **using** *strategy-conforming-path-exists-single assms* **by** *blast*

then interpret *vmc-path* $G \ P \ v \ p \ \sigma$.

have $P = LCons \ v \ LNil$ **using** $P\text{-deadend-}v0\text{-}LCons \ \langle \text{deadend } v \rangle$ **by** *blast*

hence $\neg \text{winning-path } p \ P$ **unfolding** *winning-path-def* **using** $\langle v \in VV \ p \rangle$ **by** *auto*

thus *?thesis* **using** *winning-strategy-def vmc-path-axioms* **by** *blast*

qed

lemma *winning-strategy-on-deadends*:

```

  assumes  $v \in VV$   $p$  deadend  $v$  strategy  $p$   $\sigma$ 
  shows winning-strategy  $p^{**}$   $\sigma$   $v$ 
proof
  fix  $P$  assume vmc-path  $G$   $P$   $v$   $p^{**}$   $\sigma$ 
  then interpret vmc-path  $G$   $P$   $v$   $p^{**}$   $\sigma$  .
  have  $P = LCons\ v\ LNil$  using  $P$ -deadend- $v0$ - $LCons$   $\langle$ deadend  $v$  $\rangle$  by blast
  thus winning-path  $p^{**}$   $P$  unfolding winning-path-def
    using  $\langle v \in VV\ p \rangle$   $P$ -valid paths-are-winning-for-one-player by auto
qed

```

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

```

  fix  $P$  assume vmc-path  $G$   $P$   $(\sigma\ v0)$   $p$   $\sigma$ 
  then interpret vmc-path  $G$   $P$   $\sigma\ v0$   $p$   $\sigma$  .
  have  $v0 \rightarrow \sigma\ v0$  using  $v0$   $\sigma(1)$  strategy-def by blast
  hence winning-path  $p$  ( $LCons\ v0\ P$ )
    using  $\sigma(2)$  extension-valid-maximal-conforming winning-strategy-def by blast
  thus winning-path  $p$   $P$  using winning-path-ltl[ $of\ p\ LCons\ v0\ P$ ] by simp

```

qed

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

```

  fix  $P$  assume vmc-path  $G$   $P$   $w0$   $p$   $\sigma$ 
  then interpret vmc-path  $G$   $P$   $w0$   $p$   $\sigma$  .
  have winning-path  $p$  ( $LCons\ v0\ P$ )
    using extension-valid-maximal-conforming  $VV$ -impl1  $\sigma\ v0$  winning-strategy-def
    by auto
  thus winning-path  $p$   $P$  using winning-path-ltl[ $of\ p\ LCons\ v0\ P$ ] by auto

```

qed

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

```

  fix  $P$  assume vmc-path  $G$   $P$   $v0$   $p$   $\sigma$ 
  then interpret vmc-path  $G$   $P$   $v0$   $p$   $\sigma$  .
  show winning-path  $p$   $P$  proof (cases)
    assume deadend  $v0$ 
    thus ?thesis using  $P$ -deadend- $v0$ - $LCons$  winning-path-def  $v0$  by auto

```

next

```

  assume  $\neg$ deadend  $v0$ 
  then interpret vmc-path-no-deadend  $G$   $P$   $v0$   $p$   $\sigma$  by unfold-locales
  interpret ltlP: vmc-path  $G$  ltl  $P$   $w0$   $p$   $\sigma$  using vmc-path-ltl .

```

```

    have winning-path p (ltl P)
    using  $\sigma(2)$  v0-edge-w0 vmc-path-ltl winning-strategy-def by blast
    thus winning-path p P
    using winning-path-LCons by (metis P-LCons' ltlP.P-LCons ltlP.P-not-null)
qed
qed

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
  fix P assume vmc-path G P v0 p  $\sigma$ 
  then interpret vmc-path G P v0 p  $\sigma$  .
  have  $\neg$ deadend v0 using  $\langle v0 \rightarrow w \rangle$  by blast
  then interpret vmc-path-no-deadend G P v0 p  $\sigma$  by unfold-locales
  have winning-path p (ltl P)
  using  $\sigma(2)[\text{unfolded winning-strategy-def}]$  v0(1,2) v0-conforms vmc-path-ltl by presburger
  thus winning-path p P using winning-path-LCons by (metis P-LCons Ptl-not-null)
qed

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 set
  and p :: Player
  — The set of good strategies on a node v
  and good :: 'a  $\Rightarrow$  'a Strategy set
  and r :: ('a Strategy  $\times$  'a Strategy) set
  assumes S-V:  $S \subseteq V$ 
  — r is a wellorder on the set of all strategies which are good somewhere.
  and r-wo: well-order-on  $\{\sigma. \exists v \in S. \sigma \in \text{good } v\}$  r
  — Every node has a good strategy.
  and good-ex:  $\bigwedge v. v \in S \implies \exists \sigma. \sigma \in \text{good } v$ 

```

— good strategies are well-formed strategies.
and *good-strategies*: $\bigwedge v \sigma. \sigma \in \text{good } v \implies \text{strategy } p \sigma$
 — A good strategy on v is also good on possible successors of v .
and *strategies-continue*: $\bigwedge v w \sigma. \llbracket v \in S; v \rightarrow w; v \in VV \ p \implies \sigma \ v = w; \sigma \in \text{good } v \rrbracket \implies \sigma \in \text{good } w$
begin

The set of all strategies which are good somewhere.

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

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

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

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-refl* [*simp*]: *refl-on Strategies r*

using *r-wo* **unfolding** *well-order-on-def linear-order-on-def partial-order-on-def preorder-on-def*
by *blast*

lemma *r-total* [*simp*]: *total-on Strategies r*

using *r-wo* **unfolding** *well-order-on-def linear-order-on-def* **by** *blast*

lemma *r-trans* [*simp*]: *trans r*

using *r-wo* **unfolding** *well-order-on-def linear-order-on-def partial-order-on-def preorder-on-def*
by *blast*

lemma *r-wf* [*simp*]: *wf (r - Id)*

using *well-order-on-def r-wo* **by** *blast*

choose always chooses a minimal good strategy on S .

lemma *choose-works*:

assumes $v \in S$

shows *minimal-good-strategy v (choose v)*

proof—

have *wf*: *wf (r - Id)* **using** *well-order-on-def r-wo* **by** *blast*

obtain σ **where** $\sigma 1$: *minimal-good-strategy v σ*

unfolding *minimal-good-strategy-def* **by** (*meson good-ex[OF v ∈ S] wf wf-eq-minimal*)

hence σ : $\sigma \in \text{good } v \wedge \sigma'. (\sigma', \sigma) \in r - \text{Id} \implies \sigma' \notin \text{good } v$

unfolding *minimal-good-strategy-def* **by** *auto*

{ fix σ' **assume** *minimal-good-strategy v σ'*

hence σ' : $\sigma' \in \text{good } v \wedge \sigma. (\sigma, \sigma') \in r - \text{Id} \implies \sigma \notin \text{good } v$

unfolding *minimal-good-strategy-def* **by** *auto*

have $(\sigma, \sigma') \notin r - \text{Id}$ **using** $\sigma(1) \sigma'(2)$ **by** *blast*

```

    moreover have  $(\sigma', \sigma) \notin r - Id$  using  $\sigma(2) \sigma'(1)$  by auto
    moreover have  $\sigma \in Strategies$  using  $\sigma(1) \langle v \in S \rangle$  by auto
    moreover have  $\sigma' \in Strategies$  using  $\sigma'(1) \langle v \in S \rangle$  by auto
    ultimately have  $\sigma' = \sigma$ 
      using r-wo Linear-order-in-diff-Id well-order-on-Field well-order-on-def by fastforce
  }
  with  $\sigma 1$  have  $\exists! \sigma. \text{minimal-good-strategy } v \sigma$  by blast
  thus ?thesis using theI'[of minimal-good-strategy v, folded choose-def] by blast
qed

```

```

corollary
  assumes  $v \in S$ 
  shows choose-good: choose  $v \in \text{good } v$ 
    and choose-minimal:  $\bigwedge \sigma'. (\sigma', \text{choose } v) \in r - Id \implies \sigma' \notin \text{good } v$ 
    and choose-strategy: strategy  $p (\text{choose } v)$ 
  using choose-works[OF assms, unfolded minimal-good-strategy-def] good-strategies by blast+

```

```

corollary choose-in-Strategies:  $v \in S \implies \text{choose } v \in Strategies$  using choose-good by blast

```

```

lemma well-ordered-strategy-valid: strategy  $p$  well-ordered-strategy
proof-
  {
    fix  $v$  assume  $v \in S \ v \in VV \ p \neg \text{deadend } v$ 
    moreover have strategy  $p (\text{choose } v)$ 
    using choose-works[OF  $\langle v \in S \rangle$ , unfolded minimal-good-strategy-def, THEN conjunct1] good-strategies
      by blast
    ultimately have  $v \rightarrow (\lambda v. \text{choose } v \ v)$  v using strategy-def by blast
  }
  thus ?thesis unfolding well-ordered-strategy-def using valid-strategy-updates-set by force
qed

```

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  $\text{lset } (\text{path-strategies } P) \subseteq Strategies$ 
  using path-strategies-def assms choose-in-Strategies by auto

```

```

lemma path-strategies-good:
  assumes  $\text{lset } P \subseteq S \text{ enat } n < \text{llength } P$ 
  shows  $\text{path-strategies } P \$ n \in \text{good } (P \$ n)$ 
  by (simp add: path-strategies-def assms choose-good lset-lnth-member)

```

```

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

```

lemma *path-strategies-monotone-Suc*:
assumes P : $\text{lset } P \subseteq S$ *valid-path* P *path-conforms-with-strategy* p P *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–
define P' **where** $P' = \text{ldropn } n P$
hence $\text{enat } (\text{Suc } 0) < \text{llength } P'$ **using** $P(4)$
by $(\text{metis } \text{enat-ltl-Suc } \text{ldrop-eSuc-ltl } \text{ldropn-Suc-conv-ldropn } \text{lList.disc}(2) \text{ lnull-0-llength ltl-ldropn})$
then obtain $v w Ps$ **where** vw : $P' = \text{LCons } v (\text{LCons } w Ps)$
by $(\text{metis } \text{ldropn-0 } \text{ldropn-Suc-conv-ldropn } \text{ldropn-lnull } \text{lnull-0-llength})$
moreover have $\text{lset } P' \subseteq S$ **unfolding** P' -def **using** $P(1)$ $\text{lset-ldropn-subset}[of\ n\ P]$ **by** *blast*
ultimately have $v \in S\ w \in S$ **by** *auto*
moreover have $v \rightarrow w$ **using** $\text{valid-path-edges}'[of\ v\ w\ Ps, \text{folded } vw]$ $\text{valid-path-drop}[OF\ P(2)]$
 P' -def **by** *blast*
moreover have $\text{choose } v \in \text{good } v$ **using** $\text{choose-good } \langle v \in S \rangle$ **by** *blast*
moreover have $v \in VV\ p \implies \text{choose } v\ v = w$ **proof**–
assume $v \in VV\ p$
moreover have *path-conforms-with-strategy* p P' *well-ordered-strategy*
unfolding P' -def **using** *path-conforms-with-strategy-drop* $P(3)$ **by** *blast*
ultimately have *well-ordered-strategy* $v = w$ **using** vw *path-conforms-with-strategy-start* **by**
blast
thus $\text{choose } v\ v = w$ **unfolding** *well-ordered-strategy-def* **using** $\langle v \in S \rangle$ **by** *auto*
qed
ultimately have $\text{choose } v \in \text{good } w$ **using** *strategies-continue* **by** *blast*
hence $*$: $(\text{choose } v, \text{choose } w) \notin r - \text{Id}$ **using** $\text{choose-minimal } \langle w \in S \rangle$ **by** *blast*

have $(\text{choose } w, \text{choose } v) \in r$ **proof** (*cases*)
assume $\text{choose } v = \text{choose } w$
thus $?thesis$ **using** $r\text{-refl refl-onD choose-in-Strategies}[OF\ \langle v \in S \rangle]$ **by** *fastforce*
next
assume $\text{choose } v \neq \text{choose } w$
thus $?thesis$ **using** $*\ r\text{-total choose-in-Strategies}[OF\ \langle v \in S \rangle]\ \text{choose-in-Strategies}[OF\ \langle w \in S \rangle]$
by $(\text{metis } (\text{lifting})\ \text{Linear-order-in-diff-Id } r\text{-wo well-order-on-Field well-order-on-def})$
qed
hence $(\text{path-strategies } P' \$ \text{Suc } 0, \text{path-strategies } P' \$ 0) \in r$
unfolding *path-strategies-def* **using** vw **by** *simp*
thus $?thesis$ **unfolding** *path-strategies-def* P' -def
using $\text{lnth-lmap-ldropn}[OF\ \text{Suc-llength}[OF\ P(4)],\ of\ \text{choose}]$
 $\text{lnth-lmap-ldropn-Suc}[OF\ P(4),\ of\ \text{choose}]$
by *simp*
qed

lemma *path-strategies-monotone*:
assumes P : $\text{lset } P \subseteq S$ *valid-path* P *path-conforms-with-strategy* p P *well-ordered-strategy*
 $n < m$ $\text{enat } m < \text{llength } P$
shows $(\text{path-strategies } P \$ m, \text{path-strategies } P \$ n) \in r$
using *assms* **proof** (*induct* $m - n$ *arbitrary: n m*)
case $(\text{Suc } d)$
show $?case$ **proof** (*cases*)
assume $d = 0$
thus $?thesis$ **using** *path-strategies-monotone-Suc* $[OF\ P(1,2,3)]$

```

    by (metis (no-types) Suc.hyps(2) Suc.prem(4,5) Suc-diff-Suc Suc-inject Suc-leI diff-is-0-eq
diffs0-imp-equal)
  next
    assume  $d \neq 0$ 
    have  $m \neq 0$  using Suc.hyps(2) by linarith
    then obtain  $m'$  where  $m': \text{Suc } m' = m$  using not0-implies-Suc by blast
    hence  $d = m' - n$  using Suc.hyps(2) by presburger
    moreover hence  $n < m'$  using  $\langle d \neq 0 \rangle$  by presburger
    ultimately have  $(\text{path-strategies } P \ \$ \ m', \text{path-strategies } P \ \$ \ n) \in r$ 
    using Suc.hyps(1)[of  $m' \ n$ , OF - P(1,2,3)] Suc.prem(5) dual-order.strict-trans enat-ord-simps(2)
  m'
    by blast
    thus ?thesis
    using  $m'$  path-strategies-monotone-Suc[OF P(1,2,3)] by (metis (no-types) Suc.prem(5) r-trans
trans-def)
  qed
qed simp

```

lemma *path-strategies-eventually-constant*:

assumes $\neg \text{finite } P$ **lset** $P \subseteq S$ *valid-path* P *path-conforms-with-strategy* p P *well-ordered-strategy*
shows $\exists n. \forall m \geq n. \text{path-strategies } P \ \$ \ n = \text{path-strategies } P \ \$ \ m$

proof –

```

  define  $\sigma$ -set where  $\sigma$ -set = lset (path-strategies  $P$ )
  have  $\exists \sigma. \sigma \in \sigma$ -set unfolding  $\sigma$ -set-def path-strategies-def
    using assms(1) lfinite-lmap lset-nth-member-inf by blast
  then obtain  $\sigma'$  where  $\sigma': \sigma' \in \sigma$ -set  $\wedge \tau. (\tau, \sigma') \in r - \text{Id} \implies \tau \notin \sigma$ -set
    using wfE-min[of  $r - \text{Id} - \sigma$ -set] by auto
  obtain  $n$  where  $n$ : path-strategies  $P \ \$ \ n = \sigma'$ 
    using  $\sigma'(1)$  lset-lnth[of  $\sigma'$ ] unfolding  $\sigma$ -set-def by blast

```

```

{
  fix  $m$  assume  $n \leq m$ 
  have path-strategies  $P \ \$ \ n = \text{path-strategies } P \ \$ \ m$  proof (rule ccontr)
    assume *: path-strategies  $P \ \$ \ n \neq \text{path-strategies } P \ \$ \ m$ 
    with  $\langle n \leq m \rangle$  have  $n < m$  using le-imp-less-or-eq by blast
    with path-strategies-monotone have  $(\text{path-strategies } P \ \$ \ m, \text{path-strategies } P \ \$ \ n) \in r$ 
      using assms by (simp add: infinite-small-llength)
    with * have  $(\text{path-strategies } P \ \$ \ m, \text{path-strategies } P \ \$ \ n) \in r - \text{Id}$  by simp
    with  $\sigma'(2)$   $n$  have path-strategies  $P \ \$ \ m \notin \sigma$ -set by blast
    thus False unfolding  $\sigma$ -set-def path-strategies-def
      using assms(1) lfinite-lmap lset-nth-member-inf by blast
  qed
}

```

thus ?thesis by blast

qed

8.2 Eventually One Strategy

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

lemma *path-eventually-conforms-to- σ -map- n* :


```

    assumes lset P ⊆ S valid-path P path-conforms-with-strategy p P well-ordered-strategy
    shows ∃ n. path-conforms-with-strategy p (ldropn n P) (path-strategies P $ n)
  proof (cases)
    assume lfinite P
    then obtain n where llength P = enat n using lfinite-llength-enat by blast
    hence ldropn n P = LNil by simp
    thus ?thesis by (metis path-conforms-LNil)
  next
    assume ¬lfinite P
    then obtain n where n: ∧ m. n ≤ m ⇒ path-strategies P $ n = path-strategies P $ m
      using path-strategies-eventually-constant assms by blast
    let ?σ = well-ordered-strategy
    define P' where P' = ldropn n P
    { fix v assume v ∈ lset P'
      hence v ∈ S using ⟨lset P ⊆ S⟩ P'-def in-lset-ldropnD by fastforce
      from ⟨v ∈ lset P'⟩ obtain m where m: enat m < llength P' P' $ m = v by (meson
in-lset-conv-lnth)
      hence P $ m + n = v unfolding P'-def by (simp add: ⟨¬lfinite P⟩ infinite-small-llength)
      moreover have ?σ v = choose v v unfolding well-ordered-strategy-def using ⟨v ∈ S⟩ by auto
      ultimately have ?σ v = (path-strategies P $ m + n) v
      unfolding path-strategies-def using infinite-small-llength[OF ⟨¬lfinite P⟩] by simp
      hence ?σ v = (path-strategies P $ n) v using n[of m + n] by simp
    }
    moreover have path-conforms-with-strategy p P' well-ordered-strategy
      unfolding P'-def by (simp add: assms(3) path-conforms-with-strategy-drop)
    ultimately show ?thesis
      using path-conforms-with-strategy-irrelevant-updates P'-def by blast
  qed
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 ≡ { v ∈ V. ∃ σ. strategy p σ ∧ winning-strategy p σ v }

```

```

lemma winning-regionI [intro]:

```

```

  assumes v ∈ V strategy p σ winning-strategy p σ v
  shows v ∈ winning-region p
  using assms unfolding winning-region-def by blast

```

lemma *winning-region-in-V* [simp]: *winning-region* $p \subseteq V$ **unfolding** *winning-region-def* **by** *blast*

lemma *winning-region-deadends*:

assumes $v \in VV\ p$ *deadend* v

shows $v \in \text{winning-region } p^{**}$

proof

show $v \in V$ **using** $\langle v \in VV\ p \rangle$ **by** *blast*

show *winning-strategy* p^{**} σ -arbitrary v **using** *assms winning-strategy-on-deadends* **by** *simp*

qed *simp*

9.1 Paths in Winning Regions

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

assumes σ' : *strategy* $p\ \sigma'$ *winning-strategy* $p\ \sigma'\ v0$

and σ : $\bigwedge v. v \in \text{winning-region } p \implies \sigma' v = \sigma v$

shows $\text{lset } P \subseteq \text{winning-region } p$

proof

fix x **assume** $x \in \text{lset } P$

thus $x \in \text{winning-region } p$ **using** *assms vmc-path-axioms*

proof (induct arbitrary: $v0$ rule: *lset-set-induct*)

case (*find* $P\ v0$)

interpret *vmc-path* $G\ P\ v0\ p\ \sigma$ **using** *find.premis(4)* .

show ?*case* **using** $P\text{-}v0\ \sigma'(1)\ \text{find.premis}(2)\ v0\text{-}V$ **unfolding** *winning-region-def* **by** *blast*

next

case (*step* $P\ x\ v0$)

interpret *vmc-path* $G\ P\ v0\ p\ \sigma$ **using** *step.premis(4)* .

show ?*case* **proof** (*cases*)

assume *lnull* (*ltl* P)

thus ?*thesis* **using** $P\text{-lnull-ltl-LCons}\ \text{step.hyps}(2)$ **by** *auto*

next

assume $\neg \text{lnull } (\text{ltl } P)$

then interpret *vmc-path-no-deadend* $G\ P\ v0\ p\ \sigma$ **using** $P\text{-no-deadend-}v0$ **by** *unfold-locales*

have *winning-strategy* $p\ \sigma'\ w0$ **proof** (*cases*)

assume $v0 \in VV\ p$

hence *winning-strategy* $p\ \sigma'\ (\sigma' v0)$

using *strategy-extends-VVp local.step(4) step.premis(2) v0-no-deadend* **by** *blast*

moreover have $\sigma v0 = w0$ **using** *v0-conforms* $\langle v0 \in VV\ p \rangle$ **by** *blast*

moreover have $\sigma' v0 = \sigma v0$

using $\sigma\ \text{assms}(1)\ \text{step.premis}(2)\ v0\text{-}V$ **unfolding** *winning-region-def* **by** *blast*

ultimately show ?*thesis* **by** *simp*

next

assume $v0 \notin VV\ p$

thus ?*thesis* **using** $v0\text{-}V\ \text{strategy-extends-VVpstar}\ \text{step}(4)\ \text{step.premis}(2)$ **by** *simp*

qed

thus ?*thesis* **using** *step.hyps(3) step(4) sigma vmc-path-ltl* **by** *blast*

qed

qed

qed

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

assumes σ' : *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 $\text{winning-path } p \ P$
proof–
obtain n **where** $n: \text{enat } n < \text{llength } P \ P \ \$ \ n \in \text{winning-region } p$
using P **by** $(\text{meson lset-intersect-lnth})$
define P' **where** $P' = \text{ldropn } n \ P$
then interpret $P': \text{vmc-path } G \ P' \ P \ \$ \ n \ p \ \sigma$
unfolding $P'\text{-def}$ **using** $\text{vmc-path-ldropn } n(1)$ **by** blast
have $\text{winning-strategy } p \ \sigma' (P \ \$ \ n)$ **using** $\sigma'(2) \ n(2)$ **by** blast
hence $\text{lset } P' \subseteq \text{winning-region } p$
using $P'.\text{paths-stay-in-winning-region}[OF \ \sigma'(1) - \sigma]$
by blast
hence $\bigwedge v. v \in \text{lset } P' \implies \sigma v = \sigma' v$ **using** σ **by** auto
hence $\text{path-conforms-with-strategy } p \ P' \ \sigma'$
using $\text{path-conforms-with-strategy-irrelevant-updates } P'.P\text{-conforms}$
by blast
then interpret $P': \text{vmc-path } G \ P' \ P \ \$ \ n \ p \ \sigma'$ **using** $P'.\text{conforms-to-another-strategy}$ **by** blast
have $\text{winning-path } p \ P'$ **using** $\sigma'(2) \ n(2) \ P'.\text{vmc-path-axioms winning-strategy-def}$ **by** blast
thus $\text{winning-path } p \ P$ **unfolding** $P'\text{-def}$ **using** $\text{winning-path-drop-add } n(1) \ P\text{-valid}$ **by** blast
qed

9.2 Irrelevant Updates

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

lemma $\text{winning-strategy-updates}$:

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

fix P **assume** $\text{vmc-path } G \ P \ v0 \ p \ (\sigma(v := w))$

then interpret $\text{vmc-path } G \ P \ v0 \ p \ \sigma(v := w)$.

have $\bigwedge v'. v' \in \text{winning-region } p \implies \sigma v' = (\sigma(v := w)) v'$ **using** v **by** auto

hence $v \notin \text{lset } P$ **using** $v \ \text{paths-stay-in-winning-region } \sigma$ **unfolding** $\text{winning-region-def}$ **by** blast

hence $\text{path-conforms-with-strategy } p \ P \ \sigma$

using $P\text{-conforms path-conforms-with-strategy-irrelevant'}$ **by** blast

thus $\text{winning-path } p \ P$ **using** $\text{conforms-to-another-strategy } \sigma(2) \ \text{winning-strategy-def}$ **by** blast

qed

9.3 Extending Winning Regions

lemma $\text{winning-region-extends-VVp}$:

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 (rule ccontr)

obtain σ **where** $\sigma: \text{strategy } p \ \sigma \ \text{winning-strategy } p \ \sigma \ w$

using w **unfolding** $\text{winning-region-def}$ **by** blast

let $?\sigma = \sigma(v := w)$

assume $\text{contra}: v \notin \text{winning-region } p$

moreover have $\text{strategy } p \ ?\sigma$ **using** $\text{valid-strategy-updates } \sigma(1) \ \langle v \rightarrow w \rangle$ **by** blast

moreover hence $\text{winning-strategy } p \ ?\sigma \ v$

```

    using winning-strategy-updates  $\sigma$  contra  $v$  strategy-extends-backwards- $VVp$ 
    by auto
    ultimately show False using  $\langle v \rightarrow w \rangle$  unfolding winning-region-def by auto
qed

```

Unfortunately, we cannot prove the corresponding theorem *winning-region-extends- $VVpstar$* for VVp^{**} -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—

define good where good $v = \{ \sigma. \text{strategy } p \sigma \wedge \text{strategy-attracts-via } p \sigma v S W \}$ for v

let $?G = \{ \sigma. \exists v \in S - W. \sigma \in \text{good } v \}$

obtain r where r : well-order-on $?G$ r using well-order-on by blast

interpret WellOrderedStrategies $G S - W p$ good r proof

show $S - W \subseteq V$ using $\langle S \subseteq V \rangle$ by blast

next

show $\bigwedge v. v \in S - W \implies \exists \sigma. \sigma \in \text{good } v$ unfolding good-def using strategies-ex by blast

next

show $\bigwedge v \sigma. \sigma \in \text{good } v \implies \text{strategy } p \sigma$ unfolding good-def by blast

next

fix $v w \sigma$ assume $v: v \in S - W v \rightarrow w v \in VVp \implies \sigma v = w \sigma \in \text{good } v$

hence $\sigma: \text{strategy } p \sigma \text{strategy-attracts-via } p \sigma v S W$ unfolding good-def by simp-all

hence $\text{strategy-attracts-via } p \sigma w S W$ using strategy-attracts-via-successor v by blast

thus $\sigma \in \text{good } w$ unfolding good-def using $\sigma(1)$ by blast

qed (insert r)

have S - W -no-deadends: $\bigwedge v. v \in S - W \implies \neg \text{deadend } v$

using strategy-attracts-via-no-deadends[of $S W$] strategies-ex

by (metis (no-types) Diff-iff S - V rev-subsetD)

```

{
  fix v0 assume v0 ∈ S
  fix P assume P: vmc-path G P v0 p well-ordered-strategy
  then interpret vmc-path G P v0 p well-ordered-strategy .
  have visits-via P S W proof (rule ccontr)
    assume contra: ¬visits-via P S W

  hence lset P ⊆ S - W proof (induct rule: vmc-path-lset-induction)
    case base
    show v0 ∈ S - W using ⟨v0 ∈ S⟩ contra visits-via-trivial by blast
  next
  case (step P v0)
  interpret vmc-path-no-deadend G P v0 p well-ordered-strategy using step.hyps(1) .
  have insert v0 S = S using step.hyps(2) by blast
  hence *: ¬visits-via (ltl P) S W
    using visits-via-LCons[of ltl P S W v0, folded P-LCons] step.hyps(3) by auto
  hence **: w0 ∉ W using vmc-path.visits-via-trivial[OF vmc-path-ltl] by blast
  have w0 ∈ S ∪ W proof (cases)
    assume v0 ∈ VV p
    hence well-ordered-strategy v0 = w0 using v0-conforms by blast
    hence choose v0 v0 = w0 using step.hyps(2) well-ordered-strategy-def by auto
    moreover have strategy-attracts-via p (choose v0) v0 S W
      using choose-good good-def step.hyps(2) by blast
    ultimately show ?thesis
      by (metis strategy-attracts-via-successor strategy-attracts-via-v0
        choose-strategy step.hyps(2) v0-edge-w0 w0-V)
  qed (metis DiffD1 assms(2) step.hyps(2) strategy-attracts-via-successor
    strategy-attracts-via-v0 v0-edge-w0 w0-V)
  with * ** show ?case by blast
qed

have ¬lfinite P proof
  assume lfinite P
  hence deadend (llast P) using P-maximal P-not-null maximal-ends-on-deadend by blast
  moreover have llast P ∈ S - W using ⟨lset P ⊆ S - W⟩ P-not-null ⟨lfinite P⟩ lfinite-lset
by blast
  ultimately show False using S-W-no-deadends by blast
qed

obtain n where n: path-conforms-with-strategy p (ldropn n P) (path-strategies P $ n)
  using path-eventually-conforms-to-σ-map-n[OF ⟨lset P ⊆ S - W⟩ P-valid P-conforms]
  by blast
define σ' where [simp]: σ' = path-strategies P $ n
define P' where [simp]: P' = ldropn n P
interpret vmc-path G P' lhd P' p σ'
proof
  show ¬lnull P' unfolding P'-def
    using ⟨¬lfinite P⟩ lfinite-ldropn lnull-imp-lfinite by blast
  qed (simp-all add: n)
have strategy p σ' unfolding σ'-def
  using path-strategies-strategy ⟨lset P ⊆ S - W⟩ ⟨¬lfinite P⟩ infinite-small-llength

```

```

    by blast
  moreover have strategy-attracts-via p  $\sigma'$  (lhd  $P'$ )  $S$   $W$  proof-
  have  $P \ \$ \ n \in S - W$  using  $\langle \text{lset } P \subseteq S - W \rangle \langle \neg \text{lfiniteness } P \rangle \text{lset-nth-member-inf}$  by blast
  hence  $\sigma' \in \text{good } (P \ \$ \ n)$ 
    using path-strategies-good  $\sigma'$ -def  $\langle \neg \text{lfiniteness } P \rangle \langle \text{lset } P \subseteq S - W \rangle$  by blast
  hence strategy-attracts-via p  $\sigma'$  ( $P \ \$ \ n$ )  $S$   $W$  unfolding good-def by blast
  thus ?thesis unfolding  $P'$ -def using  $P$ -0 by (simp add:  $\langle \neg \text{lfiniteness } P \rangle$  infinite-small-llength)
qed
moreover from  $\langle \text{lset } P \subseteq S - W \rangle$  have  $\text{lset } P' \subseteq S - W$ 
  unfolding  $P'$ -def using lset-ldropn-subset[of  $n$   $P$ ] by blast
ultimately show False using strategy-attracts-via-lset by blast
qed
}
thus ?thesis using well-ordered-strategy-valid by blast
qed

```

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–

define good **where** good $v = \{\sigma. \text{strategy } p \ \sigma \wedge \text{winning-strategy } p \ \sigma \ v\}$ **for** v

let $?G = \{\sigma. \exists v \in \text{winning-region } p. \sigma \in \text{good } v\}$

obtain r **where** r : well-order-on $?G \ r$ **using** well-order-on **by** blast

have no-VVp-deadends: $\bigwedge v. [\![v \in \text{winning-region } p; v \in VV \ p]\!] \implies \neg \text{deadend } v$

using no-winning-strategy-on-deadends **unfolding** winning-region-def **by** blast

interpret WellOrderedStrategies G winning-region p p good r **proof**

show $\bigwedge v. v \in \text{winning-region } p \implies \exists \sigma. \sigma \in \text{good } v$

unfolding good-def winning-region-def **by** blast

next

show $\bigwedge v \ \sigma. \sigma \in \text{good } v \implies \text{strategy } p \ \sigma$ **unfolding** good-def **by** blast

next

fix $v \ w \ \sigma$ **assume** v : $v \in \text{winning-region } p \ v \rightarrow w \ v \in VV \ p \implies \sigma \ v = w \ \sigma \in \text{good } v$

hence σ : strategy $p \ \sigma$ winning-strategy $p \ \sigma \ v$ **unfolding** good-def **by** simp-all

hence winning-strategy $p \ \sigma \ w$ **proof** (cases)

assume *: $v \in VV \ p$

hence **: $\sigma \ v = w$ **using** $v(3)$ **by** blast

have $\neg \text{deadend } v$ **using** no-VVp-deadends $\langle v \in VV \ p \rangle v(1)$ **by** blast

with * ** **show** ?thesis **using** strategy-extends-VVp σ **by** blast

next

assume $v \notin VV \ p$

thus ?thesis **using** strategy-extends-VVpstar $\sigma \ \langle v \rightarrow w \rangle$ **by** blast

qed

thus $\sigma \in \text{good } w$ **unfolding** good-def **using** $\sigma(1)$ **by** blast

qed (insert winning-region-in- V r)

{

```

fix  $v0$  assume  $v0 \in \text{winning-region } p$ 
fix  $P$  assume  $P: \text{vmc-path } G \ P \ v0 \ p \ \text{well-ordered-strategy}$ 
then interpret  $\text{vmc-path } G \ P \ v0 \ p \ \text{well-ordered-strategy}$  .

have  $\text{lset } P \subseteq \text{winning-region } p$  proof (induct rule:  $\text{vmc-path-lset-induction-simple}$ )
  case ( $\text{step } P \ v0$ )
  interpret  $\text{vmc-path-no-deadend } G \ P \ v0 \ p \ \text{well-ordered-strategy}$  using  $\text{step.hyps}(1)$  .
  { assume  $v0 \in VV \ p$ 
    hence  $\text{well-ordered-strategy } v0 = w0$  using  $v0\text{-conforms}$  by  $\text{blast}$ 
    hence  $\text{choose } v0 \ v0 = w0$  by ( $\text{simp add: step.hyps}(2) \ \text{well-ordered-strategy-def}$ )
  }
  hence  $\text{choose } v0 \in \text{good } w0$  using  $\text{strategies-continue choose-good step.hyps}(2)$  by  $\text{simp}$ 
  thus  $?case$  unfolding  $\text{good-def winning-region-def}$  using  $w0-V$  by  $\text{blast}$ 
qed ( $\text{insert } \langle v0 \in \text{winning-region } p \rangle$ )

have  $\text{winning-path } p \ P$  proof (rule  $ccontr$ )
  assume  $\text{contra: } \neg \text{winning-path } p \ P$ 

  have  $\neg \text{lfinite } P$  proof
    assume  $\text{lfinite } P$ 
    hence  $\text{deadend } (\text{llast } P)$  using  $\text{maximal-ends-on-deadend}$  by  $\text{simp}$ 
    moreover have  $\text{llast } P \in \text{winning-region } p$ 
      using  $\langle \text{lset } P \subseteq \text{winning-region } p \rangle \ P\text{-not-null } \langle \text{lfinite } P \rangle \ \text{lfinite-lset}$  by  $\text{blast}$ 
    moreover have  $\text{llast } P \in VV \ p$ 
      using  $\text{contra paths-are-winning-for-one-player } \langle \text{lfinite } P \rangle$ 
      unfolding  $\text{winning-path-def}$  by  $\text{simp}$ 
    ultimately show  $\text{False}$  using  $\text{no-VVp-deadends}$  by  $\text{blast}$ 
  qed

  obtain  $n$  where  $n: \text{path-conforms-with-strategy } p \ (\text{ldropn } n \ P) \ (\text{path-strategies } P \ \$ \ n)$ 
  using  $\text{path-eventually-conforms-to-}\sigma\text{-map-}n[\text{OF } \langle \text{lset } P \subseteq \text{winning-region } p \rangle \ P\text{-valid } P\text{-conforms}]$ 
by  $\text{blast}$ 
  define  $\sigma'$  where  $[\text{simp}]: \sigma' = \text{path-strategies } P \ \$ \ n$ 
  define  $P'$  where  $[\text{simp}]: P' = \text{ldropn } n \ P$ 
  interpret  $P': \text{vmc-path } G \ P' \ \text{lhd } P' \ p \ \sigma'$  proof
    show  $\neg \text{lnull } P'$  using  $\langle \neg \text{lfinite } P \rangle$  unfolding  $P'\text{-def}$ 
    using  $\text{lfinite-ldropn lnull-imp-lfinite}$  by  $\text{blast}$ 
  qed ( $\text{simp-all add: } n$ )
  have  $\text{strategy } p \ \sigma'$  unfolding  $\sigma'\text{-def}$ 
    using  $\text{path-strategies-strategy } \langle \text{lset } P \subseteq \text{winning-region } p \rangle \ \langle \neg \text{lfinite } P \rangle$  by  $\text{blast}$ 
  moreover have  $\text{winning-strategy } p \ \sigma' \ (\text{lhd } P')$  proof—
    have  $P \ \$ \ n \in \text{winning-region } p$ 
      using  $\langle \text{lset } P \subseteq \text{winning-region } p \rangle \ \langle \neg \text{lfinite } P \rangle \ \text{lset-nth-member-inf}$  by  $\text{blast}$ 
    hence  $\sigma' \in \text{good } (P \ \$ \ n)$ 
      using  $\text{path-strategies-good choose-good } \sigma'\text{-def } \langle \neg \text{lfinite } P \rangle \ \langle \text{lset } P \subseteq \text{winning-region } p \rangle$ 
      by  $\text{blast}$ 
    hence  $\text{winning-strategy } p \ \sigma' \ (P \ \$ \ n)$  unfolding  $\text{good-def}$  by  $\text{blast}$ 
    thus  $?thesis$ 
      unfolding  $P'\text{-def}$  using  $P-0 \ \langle \neg \text{lfinite } P \rangle$  by ( $\text{simp add: infinite-small-llength lhd-ldropn}$ )
  qed
  ultimately have  $\text{winning-path } p \ P'$  unfolding  $\text{winning-strategy-def}$ 
    using  $P'.\text{vmc-path-axioms}$  by  $\text{blast}$ 

```

```

    moreover have  $\neg \text{finite } P'$  using  $\langle \neg \text{finite } P \rangle P'$ -def by simp
    ultimately show False using contra winning-path-drop-add[OF P-valid] by auto
qed
}
thus ?thesis unfolding winning-strategy-def using well-ordered-strategy-valid by auto
qed

```

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-
  obtain  $\sigma$  where  $\sigma: \text{strategy } p\ \sigma \ \bigwedge v. v \in \text{winning-region } p \implies \text{winning-strategy } p\ \sigma\ v$ 
    using merge-winning-strategies by blast
  have winning-strategy  $p\ \sigma\ v$  using strategy-extends-backwards-VVpstar[OF  $v\ \sigma(1)$ ]  $\sigma(2)$   $w$  by
  blast
  thus ?thesis unfolding winning-region-def using  $v\ \sigma(1)$  by blast
qed

```

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  $\exists w \in V - \text{winning-region } p. v \rightarrow w$ 
  using assms winning-region-extends-VVp winning-region-extends-VVpstar by blast

```

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  $\exists \sigma. \text{strategy } p\ \sigma \wedge \text{strategy-attracts-via } p\ \sigma\ v0\ (\text{insert } v0\ S)\ W$ 
proof-
  from  $v0(1,2)$  obtain  $w$  where  $v0 \rightarrow w\ w \in S$  using directly-attracted-def by blast
  from  $\langle w \in S \rangle\ \sigma(2)$  have strategy-attracts-via  $p\ \sigma\ w\ S\ W$  unfolding strategy-attracts-def by blast

```


let $?σ = σ(v0 := w)$ — Extend $σ$ to the new node.
have *strategy* p $?σ$ **using** $σ(1) \langle v0 \rightarrow w \rangle$ *valid-strategy-updates* **by** *blast*
moreover **have** *strategy-attracts-via* p $?σ$ $v0$ (*insert* $v0$ S) W **proof**
 fix P
 assume *vmc-path* G P $v0$ p $?σ$
 then interpret *vmc-path* G P $v0$ p $?σ$.
 have $\neg \text{deadend } v0$ **using** $\langle v0 \rightarrow w \rangle$ **by** *blast*
 then interpret *vmc-path-no-deadend* G P $v0$ p $?σ$ **by** *unfold-locales*

 define P'' **where** [*simp*]: $P'' = \text{ltl } P$
 have $\text{lhd } P'' = w$ **using** $v0(1)$ *v0-conforms w0-def* **by** *auto*
 hence *vmc-path* G P'' w p $?σ$ **using** *vmc-path-ltl* **by** (*simp add: w0-def*)

 have $*$: $v0 \notin S - W$ **using** $\langle v0 \notin S \rangle$ **by** *blast*
 have *override-on* $(σ(v0 := w))$ $σ(S - W) = ?σ$
 by (*rule ext*) (*metis * fun-upd-def override-on-def*)
 hence *strategy-attracts* p $?σ$ S W
 using *strategy-attracts-irrelevant-override*[*OF* $σ(2,1) \langle \text{strategy } p ?σ \rangle$] **by** *simp*
 hence *strategy-attracts-via* p $?σ$ w S W **unfolding** *strategy-attracts-def*
 using $\langle w \in S \rangle$ **by** *blast*
 hence *visits-via* P'' S W **unfolding** *strategy-attracts-via-def*
 using $\langle \text{vmc-path } G P'' w p ?σ \rangle$ **by** *blast*
 thus *visits-via* P (*insert* $v0$ S) W
 using *visits-via-LCons*[*of* $\text{ltl } P$ S W $v0$] *P-LCons* **by** *simp*
qed
ultimately show $?thesis$ **by** *blast*
qed

lemma *strategy-attracts-extends-VVpstar*:
 assumes $σ$: *strategy-attracts* p $σ$ S W
 and $v0$: $v0 \notin VV$ p $v0 \in \text{directly-attracted } p$ S
 shows *strategy-attracts-via* p $σ$ $v0$ (*insert* $v0$ S) W
proof
 fix P
 assume *vmc-path* G P $v0$ p $σ$
 then interpret *vmc-path* G P $v0$ p $σ$.
 have $\neg \text{deadend } v0$ **using** $v0(2)$ *directly-attracted-contains-no-deadends* **by** *blast*
 then interpret *vmc-path-no-deadend* G P $v0$ p $σ$ **by** *unfold-locales*
 have *visits-via* ($\text{ltl } P$) S W
 using *vmc-path.strategy-attractsE*[*OF* *vmc-path-ltl* $σ$] $v0$ *directly-attracted-def* **by** *simp*
 thus *visits-via* P (*insert* $v0$ S) W **using** *visits-via-LCons*[*of* $\text{ltl } P$ S W $v0$] *P-LCons* **by** *simp*
qed

lemma *attractor-has-strategy-single*:
 assumes $W \subseteq V$
 and $v0\text{-def}$: $v0 \in \text{attractor } p$ W (**is** $- \in ?A$)
 shows $\exists σ. \text{strategy } p$ $σ \wedge \text{strategy-attracts-via } p$ $σ$ $v0$ $?A$ W
using *assms* **proof** (*induct arbitrary: v0 rule: attractor-set-induction*)
 case (*step S*)
 have $v0 \in W \implies \exists σ. \text{strategy } p$ $σ \wedge \text{strategy-attracts-via } p$ $σ$ $v0$ $\{ \}$ W
 using *strategy-attracts-via-trivial valid-arbitrary-strategy* **by** *blast*
 moreover $\{$

```

assume *:  $v0 \in \text{directly-attracted } p \ S \ v0 \notin S$ 
from  $\text{assms}(1) \ \text{step.hyps}(1) \ \text{step.hyps}(2)$ 
  have  $\exists \sigma. \text{strategy } p \ \sigma \wedge \text{strategy-attracts } p \ \sigma \ S \ W$ 
  using  $\text{merge-attractor-strategies}$  by  $\text{auto}$ 
with *
  have  $\exists \sigma. \text{strategy } p \ \sigma \wedge \text{strategy-attracts-via } p \ \sigma \ v0 \ (\text{insert } v0 \ S) \ W$ 
  using  $\text{strategy-attracts-extends-VVp} \ \text{strategy-attracts-extends-VVpstar}$  by  $\text{blast}$ 
}
ultimately show  $?case$ 
  using  $\text{step.premis} \ \text{step.hyps}(2)$ 
   $\text{attractor-strategy-on-extends}[\text{of } p - v0 \ \text{insert } v0 \ S \ W \ W \cup S \cup \text{directly-attracted } p \ S]$ 
   $\text{attractor-strategy-on-extends}[\text{of } p - v0 \ S \quad W \ W \cup S \cup \text{directly-attracted } p \ S]$ 
   $\text{attractor-strategy-on-extends}[\text{of } p - v0 \ \{\} \quad W \ W \cup S \cup \text{directly-attracted } p \ S]$ 
  by  $\text{blast}$ 
next
  case  $(\text{union } M)$ 
  hence  $\exists S. S \in M \wedge v0 \in S$  by  $\text{blast}$ 
  thus  $?case$  by  $(\text{meson } \text{Union-upper} \ \text{attractor-strategy-on-extends} \ \text{union.hyps})$ 
qed

```

11.1 Existence

Prove that every attractor set has an attractor strategy.

```

theorem  $\text{attractor-has-strategy}$ :
  assumes  $W \subseteq V$ 
  shows  $\exists \sigma. \text{strategy } p \ \sigma \wedge \text{strategy-attracts } p \ \sigma \ (\text{attractor } p \ W) \ W$ 
proof–
  let  $?A = \text{attractor } p \ W$ 
  have  $?A \subseteq V$  by  $(\text{simp} \ \text{add: } \langle W \subseteq V \rangle \ \text{attractor-in-}V)$ 
  moreover
    have  $\bigwedge v. v \in ?A \implies \exists \sigma. \text{strategy } p \ \sigma \wedge \text{strategy-attracts-via } p \ \sigma \ v \ ?A \ W$ 
    using  $\langle W \subseteq V \rangle \ \text{attractor-has-strategy-single}$  by  $\text{blast}$ 
  ultimately show  $?thesis$  using  $\text{merge-attractor-strategies} \ \langle W \subseteq V \rangle$  by  $\text{blast}$ 
qed

end — context ParityGame

end

```

12 Positional Determinacy of Parity Games

```

theory  $\text{PositionalDeterminacy}$ 
imports
   $\text{Main}$ 
   $\text{AttractorStrategy}$ 
begin

context  $\text{ParityGame}$  begin

```

12.1 Induction Step

The proof of positional determinacy is by induction over the size of the finite set $\omega \preceq V$, 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*: $\bigwedge (G :: ('a, 'b) \text{ ParityGame-scheme}) v.$
 $\llbracket \text{card } (\omega_G \preceq V_G) < \text{card } (\omega \preceq V); v \in V_G;$
 $\text{ParityGame } G;$
 $\bigwedge v. v \in V_G \implies \neg \text{Digraph.deadend } G v \rrbracket$
 $\implies \exists p. v \in \text{ParityGame.winning-region } G p$
shows $\exists p. v \in \text{winning-region } p$

proof –

First, we determine the minimum priority and the player who likes it.

define *min-prio* **where** $\text{min-prio} = \text{Min } (\omega \preceq V)$
have $\exists p. \text{winning-priority } p \text{ min-prio}$ **by** *auto*
then obtain p **where** $p: \text{winning-priority } p \text{ min-prio}$ **by** *blast*

Then we define the tentative winning region of player p^{**} . The rest of the proof is to show that this is the complete winning region.

define $W1$ **where** $W1 = \text{winning-region } p^{**}$

For this, we define several more sets of nodes. First, U is the tentative winning region of player p .

define U **where** $U = V - W1$
define K **where** $K = U \cap (\omega - \{ \text{min-prio} \})$
define V' **where** $V' = U - \text{attractor } p K$

define G' **where** $[simp]: G' = \text{subgame } V'$
interpret G' : *ParityGame* G' **using** *subgame-ParityGame* **by** *simp*

have *U-equiv*: $\bigwedge v. v \in V \implies v \in U \longleftrightarrow v \notin \text{winning-region } p^{**}$
unfolding *U-def* *W1-def* **by** *blast*

have $V' \subseteq V$ **unfolding** *U-def* *V'-def* **by** *blast*
hence $[simp]: V_{G'} = V'$ **unfolding** *G'-def* **by** *simp*

have $V_{G'} \subseteq V$ $E_{G'} \subseteq E$ $\omega_{G'} = \omega$ **unfolding** *G'-def* **by** (*simp-all add: subgame- ω*)
have $G'.VV p = V' \cap VV p$ **unfolding** *G'-def* **using** *subgame-VV* **by** *simp*

have *V-decomp*: $V = \text{attractor } p K \cup V' \cup W1$ **proof** –
have $V \subseteq \text{attractor } p K \cup V' \cup W1$
unfolding *V'-def* *U-def* **by** *blast*
moreover have $\text{attractor } p K \subseteq V$
using *attractor-in-V[of K]* **unfolding** *K-def* *U-def* **by** *blast*
ultimately show *?thesis*

unfolding $W1\text{-def}$ $\text{winning-region-def}$ **using** $\langle V' \subseteq V \rangle$ **by** blast
qed
have $G'\text{-no-deadends}$: $\bigwedge v. v \in V_{G'} \implies \neg G'.\text{deadend } v$ **proof**–
fix v **assume** $v \in V_{G'}$
hence $*$: $v \in U - \text{attractor } p \ K$ **using** $\langle V_{G'} = V' \rangle$ $V'\text{-def}$ **by** blast
moreover **hence** $\exists w \in U. v \rightarrow w$
using $\text{removing-winning-region-induces-no-deadends}[of \ v \ p**]$ $\text{no-deadends } U\text{-equiv } U\text{-def}$
by blast
moreover **have** $\bigwedge w. \llbracket v \in VV \ p**; v \rightarrow w \rrbracket \implies w \in U$
using $*$ $U\text{-equiv winning-region-extends-}VVp$ **by** blast
ultimately **have** $\exists w \in V'. v \rightarrow w$
using $U\text{-equiv winning-region-extends-}VVp$ $\text{removing-attractor-induces-no-deadends } V'\text{-def}$
by blast
thus $\neg G'.\text{deadend } v$ **using** $\langle v \in V_{G'} \rangle \langle V' \subseteq V \rangle$ **by** simp
qed

By definition of $W1$, we obtain a winning strategy on $W1$ for player $p**$.

obtain $\sigma \ W1$ **where** $\sigma \ W1$:
 $\text{strategy } p** \ \sigma \ W1 \ \bigwedge v. v \in W1 \implies \text{winning-strategy } p** \ \sigma \ W1 \ v$
unfolding $W1\text{-def}$ **using** $\text{merge-winning-strategies}$ **by** blast

$\{$
fix v **assume** $v \in V_{G'}$

Apply the induction hypothesis to get the winning strategy for v in G' .

have $G'\text{-winning-strategy}$: $\exists p. v \in G'.\text{winning-region } p$ **proof**–
have $\text{card } (\omega_{G'} \setminus V_{G'}) < \text{card } (\omega \setminus V)$ **proof**–
 $\{$ **assume** $\text{min-prio} \in \omega_{G'} \setminus V_{G'}$
then **obtain** v **where** $v: v \in V_{G'} \ \omega_{G'} \ v = \text{min-prio}$ **by** blast
hence $v \in \omega \setminus \{ \text{min-prio} \}$ **using** $\langle \omega_{G'} = \omega \rangle$ **by** simp
hence False **using** $V'\text{-def } K\text{-def attractor-set-base } \langle V_{G'} = V' \rangle \ v(1)$
by $(\text{metis DiffD1 DiffD2 IntI contra-subsetD})$
 $\}$
hence $\text{min-prio} \notin \omega_{G'} \setminus V_{G'}$ **by** blast
moreover **have** $\text{min-prio} \in \omega \setminus V$
unfolding min-prio-def **using** $\text{priorities-finite Min-in assms}(1)$ **by** blast
moreover **have** $\omega_{G'} \setminus V_{G'} \subseteq \omega \setminus V$ **unfolding** $G'\text{-def}$ **by** simp
ultimately **show** $?thesis$ **by** $(\text{metis priorities-finite psubsetI psubset-card-mono})$
qed
thus $?thesis$ **using** $IH[of \ G'] \ \langle v \in V_{G'} \rangle \ G'\text{-no-deadends } G'.\text{ParityGame-axioms}$ **by** blast
qed

It turns out the winning region of player $p**$ is empty, so we have a strategy for player p .

have $v \in G'.\text{winning-region } p$ **proof** (rule ccontr)
assume $\neg ?thesis$
moreover **obtain** $p' \ \sigma$ **where** $p': G'.\text{strategy } p' \ \sigma \ G'.\text{winning-strategy } p' \ \sigma \ v$
using $G'\text{-winning-strategy}$ **unfolding** $G'.\text{winning-region-def}$ **by** blast
ultimately **have** $p' \neq p$ **using** $\langle v \in V_{G'} \rangle$ **unfolding** $G'.\text{winning-region-def}$ **by** blast
hence $p' = p**$ **by** (cases p ; cases p') auto
with p' **have** $\sigma: G'.\text{strategy } p** \ \sigma \ G'.\text{winning-strategy } p** \ \sigma \ v$ **by** simp-all

```

have  $v \in \text{winning-region } p^{**}$  proof
  show  $v \in V$  using  $\langle v \in V_{G'} \rangle \langle V_{G'} \subseteq V \rangle$  by blast
  define  $\sigma'$  where  $\sigma' = \text{override-on } (\text{override-on } \sigma\text{-arbitrary } \sigma W1 \ W1) \ \sigma \ V'$ 
  thus strategy  $p^{**} \ \sigma'$ 
    using valid-strategy-updates-set-strong valid-arbitrary-strategy  $\sigma W1(1)$ 
      valid-strategy-supergame  $\sigma(1) \ G'\text{-no-deadends } \langle V_{G'} = V' \rangle$ 
    unfolding  $G'\text{-def}$  by blast
  show winning-strategy  $p^{**} \ \sigma' \ v$ 
  proof (rule winning-strategyI, rule ccontr)
    fix  $P$  assume vmc-path  $G \ P \ v \ p^{**} \ \sigma'$ 
    then interpret vmc-path  $G \ P \ v \ p^{**} \ \sigma'$  .
    assume  $\neg \text{winning-path } p^{**} \ P$ 

```

First we show that P stays in V' , because if it stays in V' , then it conforms to σ , so it must be winning for p^{**} .

```

have  $\text{lset } P \subseteq V'$  proof (induct rule: vmc-path-lset-induction-closed-subset)
  fix  $v$  assume  $v \in V' \neg \text{deadend } v \ v \in VV \ p^{**}$ 
  hence  $v \in \text{ParityGame.VV } (\text{subgame } V') \ p^{**}$  by auto
  moreover have  $\neg G'.\text{deadend } v$  using  $G'\text{-no-deadends } \langle V_{G'} = V' \rangle \langle v \in V' \rangle$  by blast
  ultimately have  $\sigma \ v \in V'$ 
    using subgame-strategy-stays-in-subgame  $p'(1) \ \langle p' = p^{**} \rangle$ 
    unfolding  $G'\text{-def}$  by blast
  thus  $\sigma' \ v \in V' \cup W1$  unfolding  $\sigma'\text{-def}$  using  $\langle v \in V' \rangle$  by simp
next
  fix  $w \ v$  assume  $v \in V' \neg \text{deadend } v \ v \in VV \ p^{****} \ v \rightarrow w$ 
  show  $w \in V' \cup W1$  proof (rule ccontr)
    assume  $w \notin V' \cup W1$ 
    hence  $w \in \text{attractor } p \ K$  using  $V\text{-decomp } \langle v \rightarrow w \rangle$  by blast
    hence  $v \in \text{attractor } p \ K$  using  $\langle v \in VV \ p^{****} \rangle \text{ attractor-set-}VVp \ \langle v \rightarrow w \rangle$  by auto
    thus False using  $\langle v \in V' \rangle \ V'\text{-def}$  by blast
  qed
next
  have  $\bigwedge v. v \in W1 \implies \sigma W1 \ v = \sigma' \ v$  unfolding  $\sigma'\text{-def } V'\text{-def } U\text{-def}$  by simp
  thus  $\text{lset } P \cap W1 = \{\}$ 
    using path-hits-winning-region-is-winning  $\sigma W1 \ \langle \neg \text{winning-path } p^{**} \ P \rangle$ 
    unfolding  $W1\text{-def}$ 
    by blast
next
  show  $v \in V'$  using  $\langle V_{G'} = V' \rangle \langle v \in V_{G'} \rangle$  by blast
qed

```

This concludes the proof of $\text{lset } P \subseteq V'$.

```

hence  $G'.\text{valid-path } P$  using subgame-valid-path by simp
moreover have  $G'.\text{maximal-path } P$ 
  using  $\langle \text{lset } P \subseteq V' \rangle \text{ subgame-maximal-path } \langle V' \subseteq V \rangle$  by simp
moreover have  $G'.\text{path-conforms-with-strategy } p^{**} \ P \ \sigma$  proof –
  have  $G'.\text{path-conforms-with-strategy } p^{**} \ P \ \sigma'$ 
    using subgame-path-conforms-with-strategy  $\langle V' \subseteq V \rangle \langle \text{lset } P \subseteq V' \rangle$ 
    by simp
  moreover have  $\bigwedge v. v \in \text{lset } P \implies \sigma' \ v = \sigma \ v$  using  $\langle \text{lset } P \subseteq V' \rangle \sigma'\text{-def}$  by auto

```

```

ultimately show ?thesis
  using G'.path-conforms-with-strategy-irrelevant-updates by blast
qed
ultimately have G'.winning-path p** P
  using  $\sigma(2)$  G'.winning-strategy-def G'.valid-maximal-conforming-path-0 P-0 P-not-null
  by blast
moreover have G'.VV p****  $\subseteq$  VV p**** using subgame-VV-subset G'-def by blast
ultimately show False
  using G'.winning-path-supergame[of p**]  $\langle \omega_{G'} = \omega \rangle$ 
   $\langle \neg \text{winning-path } p** P \rangle$  ParityGame-axioms
  by blast
qed
qed
moreover have  $v \in V$  using  $\langle V_{G'} \subseteq V \rangle \langle v \in V_{G'} \rangle$  by blast
ultimately have  $v \in W1$  unfolding W1-def winning-region-def by blast
thus False using  $\langle v \in V_{G'} \rangle$  using U-def V'-def  $\langle V_{G'} = V' \rangle \langle v \in V_{G'} \rangle$  by blast
qed
} note recursion = this

```

We compose a winning strategy for player p on $V - W1$ out of three pieces.

First, if we happen to land in the attractor region of K , we follow the attractor strategy. This is good because the priority of the nodes in K is good for player p , so he likes to go there.

```

obtain  $\sigma 1$ 
  where  $\sigma 1$ : strategy p  $\sigma 1$ 
         strategy-attracts p  $\sigma 1$  (attractor p  $K$ )  $K$ 
  using attractor-has-strategy[of  $K$  p] K-def U-def by auto

```

Next, on G' we follow the winning strategy whose existence we proved earlier.

```

have G'.winning-region p =  $V_{G'}$  using recursion unfolding G'.winning-region-def by blast
then obtain  $\sigma 2$ 
  where  $\sigma 2$ :  $\bigwedge v. v \in V_{G'} \implies G'.\text{strategy } p \sigma 2$ 
          $\bigwedge v. v \in V_{G'} \implies G'.\text{winning-strategy } p \sigma 2 v$ 
  using G'.merge-winning-strategies by blast

```

As a last option we choose an arbitrary successor but avoid entering $W1$. In particular, this defines the strategy on the set K .

```

define succ where succ v = (SOME w.  $v \rightarrow w \wedge (v \in W1 \vee w \notin W1)$ ) for v

```

Compose the three pieces.

```

define  $\sigma$  where  $\sigma = \text{override-on (override-on succ } \sigma 2 V') \sigma 1$  (attractor p  $K - K$ )

have attractor p  $K \cap W1 = \{\}$  proof (rule ccontr)
  assume attractor p  $K \cap W1 \neq \{\}$ 
  then obtain v where v:  $v \in \text{attractor } p K$   $v \in W1$  by blast
  hence  $v \in V$  using W1-def winning-region-def by blast
  obtain P where vmc2-path G P v p  $\sigma 1$   $\sigma W1$ 
    using strategy-conforming-path-exists  $\sigma W1(1)$   $\sigma 1(1)$   $\langle v \in V \rangle$  by blast
  then interpret vmc2-path G P v p  $\sigma 1$   $\sigma W1$  .

```

```

have strategy-attracts-via  $p \sigma 1 v$  (attractor  $p K$ )  $K$  using  $v(1) \sigma 1(2)$  strategy-attracts-def by
blast
hence  $\text{lset } P \cap K \neq \{\}$  using strategy-attracts-viaE visits-via-visits by blast
hence  $\neg \text{lset } P \subseteq W1$  unfolding  $K\text{-def } U\text{-def}$  by blast
thus False unfolding  $W1\text{-def}$  using comp.paths-stay-in-winning-region  $\sigma W1 v(2)$  by auto
qed

```

On specific sets, σ behaves like one of the three pieces.

```

have  $\sigma\text{-}\sigma 1$ :  $\bigwedge v. v \in \text{attractor } p K - K \implies \sigma v = \sigma 1 v$  unfolding  $\sigma\text{-def}$  by simp
have  $\sigma\text{-}\sigma 2$ :  $\bigwedge v. v \in V' \implies \sigma v = \sigma 2 v$  unfolding  $\sigma\text{-def } V'\text{-def}$  by auto
have  $\sigma\text{-}K$ :  $\bigwedge v. v \in K \cup W1 \implies \sigma v = \text{succ } v$  proof-
  fix  $v$  assume  $v: v \in K \cup W1$ 
  hence  $v \notin V'$  unfolding  $V'\text{-def } U\text{-def}$  using attractor-set-base by auto
  with  $v$  show  $\sigma v = \text{succ } v$  unfolding  $\sigma\text{-def } U\text{-def}$  using  $\langle \text{attractor } p K \cap W1 = \{\} \rangle$ 
  by (metis (mono-tags, lifting) Diff-iff IntI UnE override-on-def override-on-emptyset)
qed

```

Show that succ succeeds in avoiding entering $W1$.

```

{ fix  $v$  assume  $v: v \in VV p$ 
  hence  $\neg \text{deadend } v$  using no-deadends by blast
  have  $\exists w. v \rightarrow w \wedge (v \in W1 \vee w \notin W1)$  proof (cases)
    assume  $v \in W1$ 
    thus ?thesis using no-deadends  $\langle \neg \text{deadend } v \rangle$  by blast
  next
    assume  $v \notin W1$ 
    show ?thesis proof (rule ccontr)
      assume  $\neg(\exists w. v \rightarrow w \wedge (v \in W1 \vee w \notin W1))$ 
      hence  $\bigwedge w. v \rightarrow w \implies \text{winning-strategy } p^{**} \sigma W1 w$  using  $\sigma W1(2)$  by blast
      hence  $\text{winning-strategy } p^{**} \sigma W1 v$ 
        using strategy-extends-backwards-VVpstar  $\sigma W1(1) \langle v \in VV p \rangle$  by simp
      hence  $v \in W1$  unfolding  $W1\text{-def } \text{winning-region-def}$  using  $\sigma W1(1) \langle \neg \text{deadend } v \rangle$  by blast
      thus False using  $\langle v \notin W1 \rangle$  by blast
    qed
  qed
  hence  $v \rightarrow \text{succ } v \vee v \in W1 \vee \text{succ } v \notin W1$  unfolding succ-def
    using someI-ex[of  $\lambda w. v \rightarrow w \wedge (v \in W1 \vee w \notin W1)$ ] by blast+
} note succ-works = this

```

```

have strategy  $p \sigma$ 
proof
  fix  $v$  assume  $v: v \in VV p \neg \text{deadend } v$ 
  hence  $v \in \text{attractor } p K - K \implies v \rightarrow \sigma v$  using  $\sigma\text{-}\sigma 1 \sigma 1(1) v$  unfolding strategy-def by auto
  moreover have  $v \in V' \implies v \rightarrow \sigma v$  proof-
    assume  $v \in V'$ 
    moreover have  $v \in V_{G'}$  using  $\langle v \in V' \rangle \langle V_{G'} = V' \rangle$  by blast
    moreover have  $v \in G'.VV p$  using  $\langle G'.VV p = V' \cap VV p \rangle \langle v \in V' \rangle \langle v \in VV p \rangle$  by blast
    moreover have  $\neg \text{Digraph.deadend } G' v$  using  $G'\text{-no-deadends } \langle v \in V_{G'} \rangle$  by blast
    ultimately have  $v \rightarrow_{G'} \sigma 2 v$  using  $\sigma 2(1) G'\text{.strategy-def}[of p \sigma 2]$  by blast
    with  $\langle v \in V' \rangle$  show  $v \rightarrow \sigma v$  using  $\langle E_{G'} \subseteq E \rangle \sigma\text{-}\sigma 2$  by (metis subsetCE)
  qed
  moreover have  $v \in K \cup W1 \implies v \rightarrow \sigma v$  using succ-works(1)  $v \sigma\text{-}K$  by auto

```

moreover have $v \in V$ using $\langle v \in VV p \rangle$ by blast
 ultimately show $v \rightarrow \sigma v$ using $V\text{-decomp}$ by blast
 qed

have $\sigma\text{-attracts: strategy-attracts } p \sigma (\text{attractor } p K) K$ **proof**–
 have strategy-attracts p (override-on $\sigma \sigma 1$ (attractor $p K - K$)) (attractor $p K$) K
 using strategy-attracts-irrelevant-override $\sigma 1 \langle \text{strategy } p \sigma \rangle$ by blast
 moreover have $\sigma = \text{override-on } \sigma \sigma 1$ (attractor $p K - K$)
 by (rule ext) (simp add: override-on-def $\sigma\text{-}\sigma 1$)
 ultimately show $?thesis$ by simp
 qed

Show that σ is a winning strategy on $V - W1$.

have $\forall v \in V - W1. \text{winning-strategy } p \sigma v$ **proof** (intro ballI winning-strategyI)
 fix $v P$ assume $P: v \in V - W1 \text{ vmc-path } G P v p \sigma$
 interpret vmc-path $G P v p \sigma$ using $P(2)$.

have $\text{lset } P \subseteq V - W1$
proof (induct rule: vmc-path-lset-induction-closed-subset)
 fix v assume $v \in V - W1 \neg \text{deadend } v v \in VV p$
 show $\sigma v \in V - W1 \cup \{\}$ **proof** (rule ccontr)
 assume $\neg ?thesis$
 hence $\sigma v \in W1$
 using $\langle \text{strategy } p \sigma \rangle \langle \neg \text{deadend } v \rangle \langle v \in VV p \rangle$
 unfolding strategy-def by blast
 hence $v \notin K$ using succ-works(2)[$OF \langle v \in VV p \rangle \langle v \in V - W1 \rangle \sigma\text{-}K$] by auto
 moreover have $v \notin \text{attractor } p K - K$ **proof**
 assume $v \in \text{attractor } p K - K$
 hence $\sigma v \in \text{attractor } p K$
 using attracted-strategy-step $\langle \text{strategy } p \sigma \rangle \sigma\text{-attracts } \langle \neg \text{deadend } v \rangle \langle v \in VV p \rangle$
 attractor-set-base
 by blast
 thus False using $\langle \sigma v \in W1 \rangle \langle \text{attractor } p K \cap W1 = \{\} \rangle$ by blast
 qed
 moreover have $v \notin V'$ **proof**
 assume $v \in V'$
 have $\sigma 2 v \in V_{G'}$ **proof** (rule $G'.\text{valid-strategy-in-}V[\text{of } p \sigma 2 v]$)
 have $v \in V_{G'}$ using $\langle V_{G'} = V' \rangle \langle v \in V' \rangle$ by simp
 thus $\neg G'.\text{deadend } v$ using $G'\text{-no-deadends}$ by blast
 show $G'.\text{strategy } p \sigma 2$ using $\sigma 2(1) \langle v \in V_{G'} \rangle$ by blast
 show $v \in G'.VV p$ using $\langle v \in VV p \rangle \langle G'.VV p = V' \cap VV p \rangle \langle v \in V' \rangle$ by simp
 qed
 hence $\sigma v \in V_{G'}$ using $\langle v \in V' \rangle \sigma\text{-}\sigma 2$ by simp
 thus False using $\langle V_{G'} = V' \rangle \langle \sigma v \in W1 \rangle V'\text{-def } U\text{-def}$ by blast
 qed
 ultimately show False using $\langle v \in V - W1 \rangle V\text{-decomp}$ by blast
 qed
 next
 fix $v w$ assume $v \in V - W1 \neg \text{deadend } v v \in VV p ** v \rightarrow w$
 show $w \in V - W1 \cup \{\}$
proof (rule ccontr)
 assume $\neg ?thesis$


```

hence  $w \in W1$  using  $\langle v \rightarrow w \rangle$  by blast
let  $? \sigma = \sigma W1(v := w)$ 
have winning-strategy  $p^{**} \sigma W1 w$  using  $\langle w \in W1 \rangle \sigma W1(2)$  by blast
moreover have  $\neg(\exists \sigma. \text{strategy } p^{**} \sigma \wedge \text{winning-strategy } p^{**} \sigma v)$ 
  using  $\langle v \in V - W1 \rangle$  unfolding W1-def winning-region-def by blast
ultimately have winning-strategy  $p^{**} ? \sigma w$ 
  using winning-strategy-updates[of  $p^{**} \sigma W1 w v w$ ]  $\sigma W1(1) \langle v \rightarrow w \rangle$ 
  unfolding winning-region-def by blast
moreover have strategy  $p^{**} ? \sigma$  using  $\langle v \rightarrow w \rangle \sigma W1(1)$  valid-strategy-updates by blast
ultimately have winning-strategy  $p^{**} ? \sigma v$ 
  using strategy-extends-backwards-VVp[of  $v p^{**} ? \sigma w$ ]
     $\langle v \in VV p^{**} \rangle \langle v \rightarrow w \rangle$ 
  by auto
hence  $v \in W1$  unfolding W1-def winning-region-def
  using  $\langle \text{strategy } p^{**} ? \sigma \rangle \langle v \in V - W1 \rangle$  by blast
thus False using  $\langle v \in V - W1 \rangle$  by blast
qed
qed (insert  $P(1)$ , simp-all)

```

This concludes the proof of $\text{lset } P \subseteq V - W1$.

```

hence  $\text{lset } P \subseteq \text{attractor } p K \cup V'$  using V-decomp by blast

```

```

have  $\neg \text{lfinite } P$ 
  using no-deadends lfinite-lset maximal-ends-on-deadend[of  $P$ ] P-maximal P-not-null lset-P-V
  by blast

```

Every σ -conforming path starting in $V - W1$ is winning. We distinguish two cases:

1. P eventually stays in V' . Then P is winning because $\sigma 2$ is winning.
2. P visits K infinitely often. Then P is winning because of the priority of the nodes in K .

```

show winning-path  $p P$ 
proof (cases)
  assume  $\exists n. \text{lset } (\text{ldropn } n P) \subseteq V'$ 

```

The first case: P eventually stays in V' .

```

then obtain  $n$  where  $n: \text{lset } (\text{ldropn } n P) \subseteq V'$  by blast
define  $P'$  where  $P' = \text{ldropn } n P$ 
hence  $\text{lset } P' \subseteq V'$  using  $n$  by blast
interpret vmc-path': vmc-path  $G' P' \text{ lhd } P' p \sigma 2$  proof
  show  $\neg \text{lnull } P'$  unfolding  $P'$ -def
    using  $\langle \neg \text{lfinite } P \rangle \text{lfinite-ldropn lnull-imp-lfinite}$  by blast
  show  $G'.\text{valid-path } P'$  proof–
    have valid-path  $P'$  unfolding  $P'$ -def by simp
    thus ?thesis using subgame-valid-path  $\langle \text{lset } P' \subseteq V' \rangle G'$ -def by blast
  qed
  show  $G'.\text{maximal-path } P'$  proof–
    have maximal-path  $P'$  unfolding  $P'$ -def by simp
    thus ?thesis using subgame-maximal-path  $\langle \text{lset } P' \subseteq V' \rangle \langle V' \subseteq V \rangle G'$ -def by blast
  qed
  show  $G'.\text{path-conforms-with-strategy } p P' \sigma 2$  proof–

```

```

have path-conforms-with-strategy p P' σ unfolding P'-def by simp
hence path-conforms-with-strategy p P' σ2
  using path-conforms-with-strategy-irrelevant-updates ⟨lset P' ⊆ V'⟩ σ-σ2
  by blast
thus ?thesis
  using subgame-path-conforms-with-strategy ⟨lset P' ⊆ V'⟩ ⟨V' ⊆ V⟩ G'-def
  by blast
qed
qed simp
have G'.winning-strategy p σ2 (lhd P')
  using ⟨lset P' ⊆ V'⟩ vmc-path'.P-not-null σ2(2)[of lhd P'] ⟨VG' = V'⟩ llist.set-sel(1)
  by blast
hence G'.winning-path p P' using G'.winning-strategy-def vmc-path'.vmc-path-axioms by blast
moreover have G'.VV p** ⊆ VV p** unfolding G'-def using subgame-VV by simp
ultimately have winning-path p P'
  using G'.winning-path-supergame[of p P' G] ⟨ωG' = ω⟩ ParityGame-axioms by blast
thus ?thesis
  unfolding P'-def
  using infinite-small-llength[OF ⟨¬lfinite P⟩]
    winning-path-drop-add[of P p n] P-valid
  by blast
next
assume asm: ¬(∃ n. lset (ldropn n P) ⊆ V')

```

The second case: P visits K infinitely often. Then min-prio occurs infinitely often on P .

```

have min-prio ∈ path-inf-priorities P
unfolding path-inf-priorities-def proof (intro CollectI allI)
  fix n
  obtain k1 where k1: ldropn n P $ k1 ∉ V' using asm by (metis lset-lnth subsetI)
  define k2 where k2 = k1 + n
  interpret vmc-path G ldropn k2 P P $ k2 p σ
  using vmc-path-ldropn infinite-small-llength ⟨¬lfinite P⟩ by blast
  have P $ k2 ∉ V' unfolding k2-def
  using k1 lnth-ldropn infinite-small-llength[OF ⟨¬lfinite P⟩] by simp
  hence P $ k2 ∈ attractor p K using ⟨¬lfinite P⟩ ⟨lset P ⊆ V - W1⟩
  by (metis DiffI U-def V'-def lset-nth-member-inf)
  then obtain k3 where k3: ldropn k2 P $ k3 ∈ K
  using σ-attracts strategy-attractsE unfolding G'.visits-via-def by blast
  define k4 where k4 = k3 + k2
  hence P $ k4 ∈ K
  using k3 lnth-ldropn infinite-small-llength[OF ⟨¬lfinite P⟩] by simp
  moreover have k4 ≥ n unfolding k4-def k2-def
  using le-add2 le-trans by blast
  moreover have ldropn n P $ k4 - n = P $ (k4 - n) + n
  using lnth-ldropn infinite-small-llength ⟨¬lfinite P⟩ by blast
  ultimately have ldropn n P $ k4 - n ∈ K by simp
  hence lset (ldropn n P) ∩ K ≠ {}
  using ⟨¬lfinite P⟩ lfinite-ldropn in-lset-conv-lnth[of ldropn n P $ k4 - n]
  by blast
  thus min-prio ∈ lset (ldropn n (lmap ω P)) unfolding K-def by auto
qed
thus ?thesis unfolding winning-path-def

```

```

    using path-inf-priorities-at-least-min-prio[OF P-valid, folded min-prio-def]
      ⟨winning-priority p min-prio⟩ ⟨¬lfinite P⟩
    by blast
  qed
qed
hence  $\forall v \in V. \exists p \sigma. \text{strategy } p \sigma \wedge \text{winning-strategy } p \sigma v$ 
  unfolding W1-def winning-region-def using ⟨strategy p σ⟩ by blast
hence  $\exists p \sigma. \text{strategy } p \sigma \wedge \text{winning-strategy } p \sigma v$  using ⟨v ∈ V⟩ by simp
thus ?thesis unfolding winning-region-def using ⟨v ∈ V⟩ by blast
qed

```

12.2 Positional Determinacy without Deadends

theorem *positional-strategy-exists-without-deadends:*

```

  assumes  $v \in V \wedge v. v \in V \implies \neg \text{deadend } v$ 
  shows  $\exists p. v \in \text{winning-region } p$ 
  using assms ParityGame-axioms
  by (induct card (ω ‘ V) arbitrary: G v rule: nat-less-induct)
    (rule ParityGame.positional-strategy-induction-step, simp-all)

```

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-
  { fix p
    define A where  $A = \text{attractor } p (\text{deadends } p^{**})$ 
    assume v0-in-attractor:  $v0 \in \text{attractor } p (\text{deadends } p^{**})$ 
    then obtain σ where  $\sigma: \text{strategy } p \sigma \text{ strategy-attracts } p \sigma A (\text{deadends } p^{**})$ 
      using attractor-has-strategy[of deadends p** p] A-def deadends-in-V by blast

    have  $A \subseteq V$  using A-def using attractor-in-V deadends-in-V by blast
    hence  $A - \text{deadends } p^{**} \subseteq V$  by auto

    have winning-strategy p σ v0 proof (unfold winning-strategy-def, intro allI impI)
      fix P assume vmc-path G P v0 p σ
      then interpret vmc-path G P v0 p σ .
      show winning-path p P
        using visits-deadend[of p**] σ(2) strategy-attracts-lset v0-in-attractor
        unfolding A-def by simp
    qed
    hence  $\exists p \sigma. \text{strategy } p \sigma \wedge \text{winning-strategy } p \sigma v0$  using σ by blast
  } note lemma-path-to-deadend = this
  define A where  $A \text{ p} = \text{attractor } p (\text{deadends } p^{**})$  for p

```

Remove the attractor sets of the sets of deadends.

```

  define V' where  $V' = V - A \text{ Even} - A \text{ Odd}$ 
  hence  $V' \subseteq V$  by blast
  show ?thesis proof (cases)

```

assume $v0 \in V'$
 define G' where $G' = \text{subgame } V'$
 interpret G' : $\text{ParityGame } G'$ **unfolding** G' -def **using** $\text{subgame-ParityGame}$.
 have $V_{G'} = V'$ **unfolding** G' -def **using** $\langle V' \subseteq V \rangle$ **by** *simp*
 hence $v0 \in V_{G'}$ **using** $\langle v0 \in V' \rangle$ **by** *simp*
 moreover have V' -no-deadends: $\bigwedge v. v \in V_{G'} \implies \neg G'.\text{deadend } v$ **proof** –
 fix v assume $v \in V_{G'}$
 moreover have $V' = V - A \text{ Even} - A \text{ Even}^{**}$ **using** V' -def **by** *simp*
 ultimately show $\neg G'.\text{deadend } v$
 using $\text{subgame-without-deadends}$ $\langle v \in V_{G'} \rangle$ **unfolding** A -def G' -def **by** *blast*
 qed
 ultimately obtain $p \sigma$ where σ : $G'.\text{strategy } p \sigma$ $G'.\text{winning-strategy } p \sigma$ $v0$
 using $G'.\text{positional-strategy-exists-without-deadends}$
 unfolding $G'.\text{winning-region-def}$
 by *blast*

 have V' -no-deadends': $\bigwedge v. v \in V' \implies \neg \text{deadend } v$ **proof** –
 fix v assume $v \in V'$
 hence $\neg G'.\text{deadend } v$ **using** V' -no-deadends $\langle V' \subseteq V \rangle$ **unfolding** G' -def **by** *auto*
 thus $\neg \text{deadend } v$ **unfolding** G' -def **using** $\langle V' \subseteq V \rangle$ **by** *auto*
 qed

 obtain $\sigma\text{-attr}$
 where $\sigma\text{-attr}$: $\text{strategy } p \sigma\text{-attr}$ $\text{strategy-attracts } p \sigma\text{-attr} (A \ p)$ ($\text{deadends } p^{**}$)
 using $\text{attractor-has-strategy}[OF \ \text{deadends-in-}V]$ **unfolding** A -def **by** *blast*
 define σ' where $\sigma' = \text{override-on } \sigma \ \sigma\text{-attr} (A \ \text{Even} \cup A \ \text{Odd})$
 have $\sigma'\text{-is-}\sigma\text{-on-}V'$: $\bigwedge v. v \in V' \implies \sigma' v = \sigma v$
 unfolding V' -def σ' -def A -def **by** (*cases* p) *simp-all*

 have $\text{strategy } p \sigma'$ **proof** –
 have $\sigma' = \text{override-on } \sigma\text{-attr} \sigma$ ($UNIV - A \ \text{Even} - A \ \text{Odd}$)
 unfolding σ' -def override-on-def **by** (*rule ext*) *simp*
 moreover have $\text{strategy } p$ ($\text{override-on } \sigma\text{-attr} \sigma \ V'$)
 using $\text{valid-strategy-supergame}$ $\sigma\text{-attr}(1) \ \sigma(1) \ V'\text{-no-deadends}$ $\langle V_{G'} = V' \rangle$
 unfolding G' -def **by** *blast*
 ultimately show $?thesis$ **by** (*simp add: valid-strategy-only-in-}V \ V'\text{-def override-on-def}*)
 qed
 moreover have $\text{winning-strategy } p \sigma' v0$ **proof** (*rule winning-strategyI, rule ccontr*)
 fix P assume $\text{vmc-path } G \ P \ v0 \ p \ \sigma'$
 then interpret $\text{vmc-path } G \ P \ v0 \ p \ \sigma'$.
 interpret $\text{vmc-path-no-deadend } G \ P \ v0 \ p \ \sigma'$
 using $V'\text{-no-deadends'}$ $\langle v0 \in V' \rangle$ **by** *unfold-locales*
 assume *contra*: $\neg \text{winning-path } p \ P$

 have $\text{lset } P \subseteq V'$ **proof** (*induct rule: vmc-path-lset-induction-closed-subset*)
 fix v assume $v \in V' \neg \text{deadend } v \ v \in VV \ p$
 hence $v \in G'.VV \ p$ **unfolding** G' -def **by** (*simp add: }v \in V'}*)
 moreover have $\neg G'.\text{deadend } v$ **using** $V'\text{-no-deadends}$ $\langle v \in V' \rangle \ \langle V_{G'} = V' \rangle$ **by** *blast*
 moreover have $G'.\text{strategy } p \ \sigma'$
 using $G'.\text{valid-strategy-only-in-}V \ \sigma'\text{-def } \sigma'\text{-is-}\sigma\text{-on-}V' \ \sigma(1) \ \langle V_{G'} = V' \rangle$ **by** *auto*
 ultimately show $\sigma' v \in V' \cup A \ p$ **using** $\text{subgame-strategy-stays-in-subgame}$
 unfolding G' -def **by** *blast*

```

next
  fix  $v\ w$  assume  $v \in V' \neg \text{deadend } v\ v \in VV\ p^{**}\ v \rightarrow w$ 
  have  $w \notin A\ p^{**}$  proof
    assume  $w \in A\ p^{**}$ 
    hence  $v \in A\ p^{**}$  unfolding  $A\text{-def}$ 
    using  $\langle v \in VV\ p^{**} \rangle\ \langle v \rightarrow w \rangle\ \text{attractor-set-}VVp$  by blast
    thus False using  $\langle v \in V' \rangle\ \text{unfolding } V'\text{-def}$  by  $(\text{cases } p)\ \text{auto}$ 
  qed
  thus  $w \in V' \cup A\ p$  unfolding  $V'\text{-def}$  using  $\langle v \rightarrow w \rangle$  by  $(\text{cases } p)\ \text{auto}$ 
next
  show  $\text{lset } P \cap A\ p = \{\}$  proof (rule ccontr)
    assume  $\text{lset } P \cap A\ p \neq \{\}$ 
    have strategy-attracts  $p$  (override-on  $\sigma'\ \sigma\text{-attr}$  ( $A\ p - \text{deadends } p^{**}$ ))
      ( $A\ p$ )
      (deadends  $p^{**}$ )
    using strategy-attracts-irrelevant-override[OF  $\sigma\text{-attr}(2)\ \sigma\text{-attr}(1)\ \langle \text{strategy } p\ \sigma' \rangle$ ]
    by blast
    moreover have override-on  $\sigma'\ \sigma\text{-attr}$  ( $A\ p - \text{deadends } p^{**}$ ) =  $\sigma'$ 
      by (rule ext, unfold  $\sigma'\text{-def}$ , cases  $p$ ) (simp-all add: override-on-def)
    ultimately have strategy-attracts  $p\ \sigma'$  ( $A\ p$ ) (deadends  $p^{**}$ ) by simp
    hence  $\text{lset } P \cap \text{deadends } p^{**} \neq \{\}$ 
    using  $\langle \text{lset } P \cap A\ p \neq \{\} \rangle\ \text{attracted-path}[OF\ \text{deadends-in-}V]$  by simp
    thus False using contra visits-deadend[of  $p^{**}$ ] by simp
  qed
  qed (insert  $\langle v0 \in V' \rangle$ )

  then interpret vmc-path  $G'\ P\ v0\ p\ \sigma'$ 
    unfolding  $G'\text{-def}$  using subgame-path-vmc-path[OF  $\langle V' \subseteq V \rangle$ ] by blast
  have  $G'.\text{path-conforms-with-strategy } p\ P\ \sigma$  proof–
    have  $\bigwedge v. v \in \text{lset } P \implies \sigma' v = \sigma v$ 
      using  $\sigma'\text{-is-}\sigma\text{-on-}V'\ \langle V_{G'} = V' \rangle\ \text{lset-}P\text{-}V$  by blast
    thus  $G'.\text{path-conforms-with-strategy } p\ P\ \sigma$ 
      using  $P\text{-conforms } G'.\text{path-conforms-with-strategy-irrelevant-updates}$  by blast
  qed
  then interpret vmc-path  $G'\ P\ v0\ p\ \sigma$  using conforms-to-another-strategy by blast
  have  $G'.\text{winning-path } p\ P$ 
    using  $\sigma(2)[\text{unfolded } G'.\text{winning-strategy-def}]\ \text{vmc-path-axioms}$  by blast
  from  $\neg \text{winning-path } p\ P$ 
     $G'.\text{winning-path-supergame}[OF\ \text{this ParityGame-axioms, unfolded } G'\text{-def}]$ 
    subgame-}VV\text{-subset}[of }p^{**}\ V^{\uparrow}]
    subgame-}\omega[of }V^{\uparrow}]
    show False by blast
  qed
  ultimately show ?thesis unfolding winning-region-def using  $\langle v0 \in V \rangle$  by blast
next
  assume  $v0 \notin V'$ 
  then obtain  $p$  where  $v0 \in \text{attractor } p\ (\text{deadends } p^{**})$ 
    unfolding  $V'\text{-def } A\text{-def}$  using  $\langle v0 \in V \rangle$  by blast
  thus ?thesis unfolding winning-region-def
    using lemma-path-to-deadend  $\langle v0 \in V \rangle$  by blast
  qed
  qed

```

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 $V = \text{winning-region Even} \cup \text{winning-region Odd}$

and $\text{winning-region Even} \cap \text{winning-region Odd} = \{\}$

proof

show $V \subseteq \text{winning-region Even} \cup \text{winning-region Odd}$

by (rule subsetI) (metis (full-types) Un-iff other-other-player positional-strategy-exists)

next

show $\text{winning-region Even} \cup \text{winning-region Odd} \subseteq V$

by (rule subsetI) (meson Un-iff subsetCE winning-region-in-V)

next

show $\text{winning-region Even} \cap \text{winning-region Odd} = \{\}$

using winning-strategy-only-for-one-player[of Even]

unfolding winning-region-def **by** auto

qed

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* :: $\text{Player} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$

for $p :: \text{Player}$ **and** $W :: 'a \text{ set}$ **where**

Base [intro!]: $v \in W \implies v \in \text{attractor-inductive } p \ W$

| VVp : $\llbracket v \in VV \ p; \exists w. v \rightarrow w \wedge w \in \text{attractor-inductive } p \ W \rrbracket$
 $\implies v \in \text{attractor-inductive } p \ W$

| $VVpstar$: $\llbracket v \in VV \ p^{**}; \neg \text{deadend } v; \forall w. v \rightarrow w \longrightarrow w \in \text{attractor-inductive } p \ W \rrbracket$
 $\implies v \in \text{attractor-inductive } p \ W$

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  $\text{attractor-inductive } p \ W = \text{attractor } p \ W$ 
proof
  show  $\text{attractor-inductive } p \ W \subseteq \text{attractor } p \ W$  proof
    fix  $v$  assume  $v \in \text{attractor-inductive } p \ W$ 
    thus  $v \in \text{attractor } p \ W$  proof (induct rule: attractor-inductive.induct)
      case (Base  $v$ ) thus ?case using attractor-set-base by auto
    next
      case (VVp  $v$ ) thus ?case using attractor-set-VVp by auto
    next
      case (VVpstar  $v$ ) thus ?case using attractor-set-VVpstar by auto
    qed
  qed
  show  $\text{attractor } p \ W \subseteq \text{attractor-inductive } p \ W$ 
  proof –
    define  $P$  where  $P \ S \longleftrightarrow S \subseteq \text{attractor-inductive } p \ W$  for  $S$ 
    from  $\langle W \subseteq V \rangle$  have  $P \ (\text{attractor } p \ W)$  proof (induct rule: attractor-set-induction)
      case (step  $S$ )
      hence  $S \subseteq \text{attractor-inductive } p \ W$  using P-def by simp
      have  $W \cup S \cup \text{directly-attracted } p \ S \subseteq \text{attractor-inductive } p \ W$  proof
        fix  $v$  assume  $v \in W \cup S \cup \text{directly-attracted } p \ S$ 
        moreover
        { assume  $v \in W$  hence  $v \in \text{attractor-inductive } p \ W$  by blast }
        moreover
        { assume  $v \in S$  hence  $v \in \text{attractor-inductive } p \ W$ 
          by (meson  $\langle S \subseteq \text{attractor-inductive } p \ W \rangle$  rev-subsetD) }
        moreover
        { assume  $v$ -attracted:  $v \in \text{directly-attracted } p \ S$ 
          hence  $v \in V$  using  $\langle S \subseteq V \rangle$  attractor-step-bounded-by-V by blast
          hence  $v \in \text{attractor-inductive } p \ W$  proof (cases rule: VV-cases)
            assume  $v \in VV \ p$ 
            hence  $\exists w. v \rightarrow w \wedge w \in S$  using v-attracted directly-attracted-def by blast
            hence  $\exists w. v \rightarrow w \wedge w \in \text{attractor-inductive } p \ W$ 
              using  $\langle S \subseteq \text{attractor-inductive } p \ W \rangle$  by blast
            thus ?thesis by (simp add:  $\langle v \in VV \ p \rangle$  attractor-inductive.VVp)
          next
            assume  $v \in VV \ p^{**}$ 
            hence *:  $\forall w. v \rightarrow w \longrightarrow w \in S$  using v-attracted directly-attracted-def by blast
            have  $\neg \text{deadend } v$  using v-attracted directly-attracted-def by blast
            show ?thesis proof (rule ccontr)
              assume  $v \notin \text{attractor-inductive } p \ W$ 
              hence  $\exists w. v \rightarrow w \wedge w \notin \text{attractor-inductive } p \ W$ 
                by (metis attractor-inductive.VVpstar  $\langle v \in VV \ p^{**} \rangle$   $\langle \neg \text{deadend } v \rangle$ )
              hence  $\exists w. v \rightarrow w \wedge w \notin S$  using  $\langle S \subseteq \text{attractor-inductive } p \ W \rangle$  by (meson subsetCE)
              thus False using * by blast
            qed
          qed
        }
      }
    ultimately show  $v \in \text{attractor-inductive } p \ W$  by (meson UnE)
  qed
thus  $P \ (W \cup S \cup \text{directly-attracted } p \ S)$  using P-def by simp

```

```

    qed (simp add: P-def Sup-least)
  thus ?thesis using P-def by simp
qed
qed

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  $G \equiv$  ()
    pre-digraph.verts = Graph.verts  $G$ ,
    pre-digraph.arcs = Graph.arcs  $G$ ,
    tail = fst,
    head = snd
  ()

```

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  $G$  proof
  fix  $e$  assume *:  $e \in$  pre-digraph.arcs (to-pre-digraph  $G$ )
  show tail (to-pre-digraph  $G$ )  $e \in$  pre-digraph.verts (to-pre-digraph  $G$ )
    by (metis * edges-are-in- $V(1)$  pre-digraph.ext-inject pre-digraph.surjective prod.collapse to-pre-digraph-def)
  show head (to-pre-digraph  $G$ )  $e \in$  pre-digraph.verts (to-pre-digraph  $G$ )
    by (metis * edges-are-in- $V(2)$  pre-digraph.ext-inject pre-digraph.surjective prod.collapse to-pre-digraph-def)
qed (simp add: arc-to-ends-def to-pre-digraph-def)
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 proof–
  {
    fix v w assume (v,w)  $\in$  Efrom-pre-digraph G
    then obtain e where e: e  $\in$  pre-digraph.arcs G tail G e = v head G e = w
    unfolding from-pre-digraph-def by auto
    hence (v,w)  $\in$  Vfrom-pre-digraph G  $\times$  Vfrom-pre-digraph G
    unfolding from-pre-digraph-def by auto
  }
  thus Digraph (from-pre-digraph G) by (simp add: Digraph.intro subrelI)
qed
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)
  unfolding to-pre-digraph-def from-pre-digraph-def arcs-ends-def arc-to-ends-def by auto

```

```

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

```

```

proof–

```

```

  define iso
    where iso =  $\langle$ 
      iso-verts = id :: 'a  $\Rightarrow$  'a,
      iso-arcs = arc-to-ends G,
      iso-head = snd,
      iso-tail = fst
     $\rangle$ 

```

```

  have inj-on (iso-verts iso) (pre-digraph.verts G) unfolding iso-def by auto

```

```

  moreover have inj-on (iso-arcs iso) (pre-digraph.arcs G)

```

```

    unfolding iso-def arc-to-ends-def by (simp add: arc-to-ends-def inj-onI no-multi-arcs)

```

```

  moreover have  $\forall a \in$  pre-digraph.arcs G.

```

```

    iso-verts iso (tail G a) = iso-tail iso (iso-arcs iso a)

```

```

     $\wedge$  iso-verts iso (head G a) = iso-head iso (iso-arcs iso a)

```

```

    unfolding iso-def by (simp add: arc-to-ends-def)

```

```

  ultimately have digraph-isomorphism iso

```

```

    unfolding digraph-isomorphism-def using arc-to-ends-def wf-digraph-axioms by blast

```

```

  moreover have to-pre-digraph (from-pre-digraph G) = app-iso iso G

```

```

unfolding to-pre-digraph-def from-pre-digraph-def iso-def app-iso-def by (simp-all add: arcs-ends-def)

ultimately show ?thesis unfolding digraph-iso-def by blast
qed

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.