Positional Determinacy of Parity Games

Christoph Dittmann
christoph.dittmann@tu-berlin.de

April 17, 2016

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

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 $\omega : V \to \mathbb{N}$ 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 $\sigma : V_0 \to V$ 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 $nat \Rightarrow 'a\ option$ with the property that the domain is an initial segment of nat (and where $'a$ is the node type). However, it turned out that coinductive lists give simpler proofs.

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

Another point is that we make extensive use of locales, in particular to represent maximal paths conforming to a specific strategy. Thus proofs often start with **interpret** *vmc-path* G P v_0 p σ 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*: $x \in lset\ xs \Longrightarrow \exists n.\ lnth\ xs\ n = x$
  ⟨*proof*⟩

lemma *lset-lnth-member*: $[\![\ lset\ xs \subseteq A;\ enat\ n < llength\ xs\ ]\!] \Longrightarrow lnth\ xs\ n \in A$
  ⟨*proof*⟩

lemma *lset-nth-member-inf*: $[\![\ \neg lfinite\ xs;\ lset\ xs \subseteq A\ ]\!] \Longrightarrow lnth\ xs\ n \in A$
  ⟨*proof*⟩

lemma *lset-intersect-lnth*: $lset\ xs \cap A \neq \{\} \Longrightarrow \exists n.\ enat\ n < llength\ xs \wedge lnth\ xs\ n \in A$
  ⟨*proof*⟩

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

lemma *lfinite-lset*: $lfinite\ xs \Longrightarrow \neg lnull\ xs \Longrightarrow llast\ xs \in lset\ xs$
⟨*proof*⟩

lemma *lset-subset*: $\neg(\mathit{lset}\ xs \subseteq A) \Longrightarrow \exists\, n.\ \mathit{enat}\ n < \mathit{llength}\ xs \wedge \mathit{lnth}\ xs\ n \notin A$
⟨*proof*⟩

2.2 *llength*

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

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

lemma *infinite-small-llength* [*intro*]: $\neg\mathit{lfinite}\ xs \Longrightarrow \mathit{enat}\ n < \mathit{llength}\ xs$
⟨*proof*⟩

lemma *lnull-0-llength*: $\neg\mathit{lnull}\ xs \Longrightarrow \mathit{enat}\ 0 < \mathit{llength}\ xs$
⟨*proof*⟩

lemma *Suc-llength*: $\mathit{enat}\ (\mathit{Suc}\ n) < \mathit{llength}\ xs \Longrightarrow \mathit{enat}\ n < \mathit{llength}\ xs$
⟨*proof*⟩

2.3 *ltake*

lemma *ltake-lnth*: $\mathit{ltake}\ n\ xs = \mathit{ltake}\ n\ ys \Longrightarrow \mathit{enat}\ m < n \Longrightarrow \mathit{lnth}\ xs\ m = \mathit{lnth}\ ys\ m$
⟨*proof*⟩

lemma *lset-ltake-prefix* [*simp*]: $n \le m \Longrightarrow \mathit{lset}\ (\mathit{ltake}\ n\ xs) \subseteq \mathit{lset}\ (\mathit{ltake}\ m\ xs)$
⟨*proof*⟩

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

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

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

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

2.4 *ldropn*

lemma *ltl-ldrop*: $[\![\ \bigwedge xs.\ P\ xs \Longrightarrow P\ (\mathit{ltl}\ xs);\ P\ xs\ ]\!] \Longrightarrow P\ (\mathit{ldropn}\ n\ xs)$
⟨*proof*⟩

2.5 *lfinite*

lemma *lfinite-drop-set*: $\mathit{lfinite}\ xs \Longrightarrow \exists\, n.\ v \notin \mathit{lset}\ (\mathit{ldrop}\ n\ xs)$

⟨*proof*⟩

lemma *index-infinite-set*:
$\llbracket \neg lfinite\ x;\ lnth\ x\ m = y;\ \bigwedge i.\ lnth\ x\ i = y \Longrightarrow (\exists\, m > i.\ lnth\ x\ m = y)\ \rrbracket \Longrightarrow y \in lset\ (ldropn\ n\ x)$
⟨*proof*⟩

2.6 *lmap*

lemma *lnth-lmap-ldropn*:
$enat\ n < llength\ xs \Longrightarrow lnth\ (lmap\ f\ (ldropn\ n\ xs))\ 0 = lnth\ (lmap\ f\ xs)\ n$
⟨*proof*⟩

lemma *lnth-lmap-ldropn-Suc*:
$enat\ (Suc\ n) < llength\ xs \Longrightarrow lnth\ (lmap\ f\ (ldropn\ n\ xs))\ (Suc\ 0) = lnth\ (lmap\ f\ xs)\ (Suc\ n)$
⟨*proof*⟩

2.7 Notation

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

notation *lnth* (**infix** \$ *61*)

end

3 Parity Games

theory *ParityGame*
imports
Main
MoreCoinductiveList
begin

3.1 Basic definitions

$'a$ is the node type. Edges are pairs of nodes.

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

A path is a possibly infinite list of nodes.

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

3.2 Graphs

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

record $'a\ Graph =$
$verts :: 'a\ set\ (V_1)$
$arcs :: 'a\ Edge\ set\ (E_1)$
abbreviation $is\text{-}arc :: ('a, 'b)\ Graph\text{-}scheme \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (**infixl** $\rightarrow_1$ *60*) **where**
$v \rightarrow_G w \equiv (v,w) \in E_G$

locale *Digraph* =
  fixes G (**structure**)
  assumes *valid-edge-set*: $E \subseteq V \times V$
begin
lemma *edges-are-in-V* [*intro*]: $v{\to}w \Longrightarrow v \in V$ $v{\to}w \Longrightarrow w \in V$ ⟨*proof*⟩

A node without successors is a *deadend*.

abbreviation *deadend* :: $'a \Rightarrow bool$ **where** *deadend* $v \equiv \neg(\exists\, w \in V.\ v \to w)$

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* $\Rightarrow$ *bool* **where**
  valid-path-base: *valid-path LNil*
| *valid-path-base'*: $v \in V \Longrightarrow$ *valid-path* (*LCons v LNil*)
| *valid-path-cons*: ⟦ $v \in V$; $w \in V$; $v{\to}w$; *valid-path Ps*; ¬*lnull Ps*; *lhd Ps* = w ⟧
    $\Longrightarrow$ *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*) $\Longrightarrow$ *valid-path Ps*
  ⟨*proof*⟩
lemma *valid-path-ltl*: *valid-path P* $\Longrightarrow$ *valid-path* (*ltl P*)
  ⟨*proof*⟩
lemma *valid-path-drop*: *valid-path P* $\Longrightarrow$ *valid-path* (*ldropn n P*)
  ⟨*proof*⟩

lemma *valid-path-in-V*: **assumes** *valid-path P* **shows** *lset P* $\subseteq V$ ⟨*proof*⟩
lemma *valid-path-finite-in-V*: ⟦ *valid-path P*; *enat n* < *llength P* ⟧ $\Longrightarrow$ *P* \$ *n* $\in V$
  ⟨*proof*⟩

lemma *valid-path-edges'*: *valid-path* (*LCons v* (*LCons w Ps*)) $\Longrightarrow v{\to}w$
  ⟨*proof*⟩
lemma *valid-path-edges*:
  assumes *valid-path P enat* (*Suc n*) < *llength P*
  shows *P* \$ *n* $\to$ *P* \$ *Suc n*
⟨*proof*⟩

lemma *valid-path-coinduct* [*consumes 1*, *case-names base step*, *coinduct pred*: *valid-path*]:
  assumes *major*: *Q P*
    and *base*: $\bigwedge v\ P.$ *Q* (*LCons v LNil*) $\Longrightarrow v \in V$
    and *step*: $\bigwedge v\ w\ P.$ *Q* (*LCons v* (*LCons w P*)) $\Longrightarrow v{\to}w \wedge$ (*Q* (*LCons w P*) $\vee$ *valid-path* (*LCons w P*))
  shows *valid-path P*
⟨*proof*⟩

lemma *valid-path-no-deadends*:
  ⟦ *valid-path P*; *enat* (*Suc i*) < *llength P* ⟧ $\Longrightarrow$ ¬*deadend* (*P* \$ *i*)
  ⟨*proof*⟩

lemma *valid-path-ends-on-deadend*:
$[\![$ *valid-path P*; *enat i* $<$ *llength P*; *deadend* $(P \$ i)$ $]\!] \Longrightarrow$ *enat* $(Suc\ i) =$ *llength P*
⟨*proof*⟩

lemma *valid-path-prefix*: $[\![$ *valid-path P*; *lprefix P′ P* $]\!] \Longrightarrow$ *valid-path P′*
⟨*proof*⟩

lemma *valid-path-lappend*:
assumes *valid-path P valid-path P′* $[\![$ $\neg$*lnull P*; $\neg$*lnull P′* $]\!] \Longrightarrow$ *llast P*$\rightarrow$*lhd P′*
shows *valid-path* (*lappend P P′*)
⟨*proof*⟩

A valid path is still valid in a supergame.

lemma *valid-path-supergame*:
assumes *valid-path P* **and** *G′*: *Digraph G′* $V \subseteq V_{G'}$ $E \subseteq E_{G'}$
shows *Digraph.valid-path G′ P*
⟨*proof*⟩

3.4 Maximal Paths

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

coinductive *maximal-path* **where**
maximal-path-base: *maximal-path LNil*
| *maximal-path-base′*: *deadend v* $\Longrightarrow$ *maximal-path* (*LCons v LNil*)
| *maximal-path-cons*: $\neg$*lnull Ps* $\Longrightarrow$ *maximal-path Ps* $\Longrightarrow$ *maximal-path* (*LCons v Ps*)

lemma *maximal-no-deadend*: *maximal-path* (*LCons v Ps*) $\Longrightarrow$ $\neg$*deadend v* $\Longrightarrow$ $\neg$*lnull Ps*
⟨*proof*⟩
lemma *maximal-ltl*: *maximal-path P* $\Longrightarrow$ *maximal-path* (*ltl P*)
⟨*proof*⟩
lemma *maximal-drop*: *maximal-path P* $\Longrightarrow$ *maximal-path* (*ldropn n P*)
⟨*proof*⟩

lemma *maximal-path-lappend*:
assumes $\neg$*lnull P′ maximal-path P′*
shows *maximal-path* (*lappend P P′*)
⟨*proof*⟩

lemma *maximal-ends-on-deadend*:
assumes *maximal-path P lfinite P* $\neg$*lnull P*
shows *deadend* (*llast P*)
⟨*proof*⟩

lemma *maximal-ends-on-deadend′*: $[\![$ *lfinite P*; *deadend* (*llast P*) $]\!] \Longrightarrow$ *maximal-path P*
⟨*proof*⟩

lemma *infinite-path-is-maximal*: $[\![$ *valid-path P*; $\neg$*lfinite P* $]\!] \Longrightarrow$ *maximal-path P*
⟨*proof*⟩

end — locale Digraph

3.5 Parity Games

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

datatype *Player* = *Even* | *Odd*

abbreviation *other-player* $p \equiv$ (*if* p = *Even then Odd else Even*)
notation *other-player* ((-**) [*1000*] *1000*)
lemma *other-other-player* [*simp*]: p**** = p ⟨*proof*⟩

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 $V \to \mathbb{N}$ with finite image.

record *'a ParityGame* = *'a Graph* +
  player0 :: *'a set* (*V0*ı)
  priority :: *'a* ⇒ *nat* (ωı)

locale *ParityGame* = *Digraph G* **for** *G* :: (*'a*, *'b*) *ParityGame-scheme* (**structure**) +
  assumes *valid-player0-set*: $V0 \subseteq V$
    and *priorities-finite*: *finite* (ω ‘ *V*)
begin

VV p is the set of nodes belonging to player *p*.

abbreviation *VV* :: *Player* ⇒ *'a set* **where** *VV p* ≡ (*if p* = *Even then V0 else V* − *V0*)
lemma *VVp-to-V* [*intro*]: $v \in VV\ p \Longrightarrow v \in V$ ⟨*proof*⟩
lemma *VV-impl1*: $v \in VV\ p \Longrightarrow v \notin VV\ p$** ⟨*proof*⟩
lemma *VV-impl2*: $v \in VV\ p$** $\Longrightarrow v \notin VV\ p$ ⟨*proof*⟩
lemma *VV-equivalence* [*iff*]: $v \in V \Longrightarrow v \notin VV\ p \longleftrightarrow v \in VV\ p$** ⟨*proof*⟩
lemma *VV-cases* [*consumes 1*]: ⟦ $v \in V$; $v \in VV\ p \Longrightarrow P$; $v \in VV\ p$** $\Longrightarrow P$ ⟧ $\Longrightarrow P$ ⟨*proof*⟩

3.6 Sets of Deadends

definition *deadends p* ≡ {$v \in VV\ p$. *deadend v*}
lemma *deadends-in-V*: *deadends p* ⊆ *V* ⟨*proof*⟩

3.7 Subgames

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

definition *subgame* **where**
  subgame V' ≡ *G*⦇
    verts := $V \cap V'$,
    arcs := $E \cap (V' \times V')$,
    player0 := $V0 \cap V'$ ⦈

lemma *subgame-V* [*simp*]: $V_{subgame\ V'} \subseteq V$
  and *subgame-E* [*simp*]: $E_{subgame\ V'} \subseteq E$
  and *subgame-ω*: $\omega_{subgame\ V'} = \omega$
  ⟨*proof*⟩

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

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

lemma *subgame-VV* [*simp*]: *ParityGame.VV* (*subgame* V') $p = V' \cap VV\ p$ ⟨*proof*⟩
corollary *subgame-VV-subset* [*simp*]: *ParityGame.VV* (*subgame* V') $p \subseteq VV\ p$ ⟨*proof*⟩

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

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

lemma *subgame-Digraph*: *Digraph* (*subgame* V')
⟨*proof*⟩

lemma *subgame-ParityGame*:
shows *ParityGame* (*subgame* V')
⟨*proof*⟩

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

lemma *subgame-maximal-path*:
assumes V': $V' \subseteq V$ **and** P: *maximal-path* P *lset* $P \subseteq V'$
shows *Digraph.maximal-path* (*subgame* V') P
⟨*proof*⟩

3.8 Priorities Occurring Infinitely Often

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

definition *path-inf-priorities* :: *'a Path* ⇒ *nat set* **where**
path-inf-priorities $P \equiv \{k.\ \forall n.\ k \in$ *lset* (*ldropn* n (*lmap* ω 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 ¬*lfinite* P
shows $\exists k.\ k \in$ *path-inf-priorities* P
⟨*proof*⟩

lemma *path-inf-priorities-at-least-min-prio*:
assumes P: *valid-path* P **and** a: $a \in$ *path-inf-priorities* P
shows *Min* (ω ‘ V) $\leq a$
⟨*proof*⟩

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

corollary *path-inf-priorities-ltl*: *path-inf-priorities P* = *path-inf-priorities* (*ltl P*)
⟨*proof*⟩

3.9 Winning Condition

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

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

abbreviation *winning-priority p* ≡ (*if p* = *Even then even else odd*)

definition *winning-path* :: *Player* ⇒ *'a Path* ⇒ *bool* **where**
winning-path p P ≡
(¬*lfinite P* ∧ (∃ *a* ∈ *path-inf-priorities P*.
(∀ *b* ∈ *path-inf-priorities P*. *a* ≤ *b*) ∧ *winning-priority p a*))
∨ (¬*lnull P* ∧ *lfinite P* ∧ *llast P* ∈ *VV p***)
∨ (*lnull P* ∧ *p* = *Even*)

Every path has a unique winner.

lemma *paths-are-winning-for-one-player*:
assumes *valid-path P*
shows *winning-path p P* ⟷ ¬*winning-path p*** *P*
⟨*proof*⟩

lemma *winning-path-ltl*:
assumes *P*: *winning-path p P* ¬*lnull P* ¬*lnull* (*ltl P*)
shows *winning-path p* (*ltl P*)
⟨*proof*⟩

corollary *winning-path-drop*:
assumes *winning-path p P enat n* < *llength P*
shows *winning-path p* (*ldropn n P*)
⟨*proof*⟩

corollary *winning-path-drop-add*:
assumes *valid-path P winning-path p* (*ldropn n P*) *enat n* < *llength P*
shows *winning-path p P*
⟨*proof*⟩

lemma *winning-path-LCons*:
assumes *P*: *winning-path p P* ¬*lnull P*
shows *winning-path p* (*LCons v P*)
⟨*proof*⟩

lemma *winning-path-supergame*:
assumes *winning-path p P*
and *G'*: *ParityGame G' VV p*** ⊆ *ParityGame.VV G' p*** $\omega = \omega_{G'}$
shows *ParityGame.winning-path G' p P*
⟨*proof*⟩

end — locale ParityGame

3.10 Valid Maximal Paths

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

locale *vm-path* = *ParityGame* +
fixes *P v0*
assumes *P-not-null* [*simp*]: $\neg$*lnull P*
and *P-valid* [*simp*]: *valid-path P*
and *P-maximal* [*simp*]: *maximal-path P*
and *P-v0* [*simp*]: *lhd P* = *v0*
begin
lemma *P-LCons*: *P* = *LCons v0* (*ltl P*) ⟨*proof*⟩

lemma *P-len* [*simp*]: *enat 0* < *llength P* ⟨*proof*⟩
lemma *P-0* [*simp*]: *P* \$ *0* = *v0* ⟨*proof*⟩
lemma *P-lnth-Suc*: *P* \$ *Suc n* = *ltl P* \$ *n* ⟨*proof*⟩
lemma *P-no-deadends*: *enat* (*Suc n*) < *llength P* ⟹ $\neg$*deadend* (*P* \$ *n*)
⟨*proof*⟩
lemma *P-no-deadend-v0*: $\neg$*lnull* (*ltl P*) ⟹ $\neg$*deadend v0*
⟨*proof*⟩
lemma *P-no-deadend-v0-llength*: *enat* (*Suc n*) < *llength P* ⟹ $\neg$*deadend v0*
⟨*proof*⟩
lemma *P-ends-on-deadend*: ⟦ *enat n* < *llength P*; *deadend* (*P* \$ *n*) ⟧ ⟹ *enat* (*Suc n*) = *llength P*
⟨*proof*⟩

lemma *P-lnull-ltl-deadend-v0*: *lnull* (*ltl P*) ⟹ *deadend v0*
⟨*proof*⟩
lemma *P-lnull-ltl-LCons*: *lnull* (*ltl P*) ⟹ *P* = *LCons v0 LNil*
⟨*proof*⟩
lemma *P-deadend-v0-LCons*: *deadend v0* ⟹ *P* = *LCons v0 LNil*
⟨*proof*⟩

lemma *Ptl-valid* [*simp*]: *valid-path* (*ltl P*) ⟨*proof*⟩
lemma *Ptl-maximal* [*simp*]: *maximal-path* (*ltl P*) ⟨*proof*⟩

lemma *Pdrop-valid* [*simp*]: *valid-path* (*ldropn n P*) ⟨*proof*⟩
lemma *Pdrop-maximal* [*simp*]: *maximal-path* (*ldropn n P*) ⟨*proof*⟩

lemma *prefix-valid* [*simp*]: *valid-path* (*ltake n P*)
⟨*proof*⟩

lemma *extension-valid* [*simp*]: *v*→*v0* ⟹ *valid-path* (*LCons v P*)
⟨*proof*⟩
lemma *extension-maximal* [*simp*]: *maximal-path* (*LCons v P*)
⟨*proof*⟩
lemma *lappend-maximal* [*simp*]: *maximal-path* (*lappend P′ P*)
⟨*proof*⟩

lemma *v0-V* [*simp*]: *v0* ∈ *V* ⟨*proof*⟩

lemma *v0-lset-P* [*simp*]: $v0 \in lset\ P$ ⟨*proof*⟩
lemma *v0-VV*: $v0 \in VV\ p \lor v0 \in VV\ p{*}{*}$ ⟨*proof*⟩
lemma *lset-P-V* [*simp*]: $lset\ P \subseteq V$ ⟨*proof*⟩
lemma *lset-ltl-P-V* [*simp*]: $lset\ (ltl\ P) \subseteq V$ ⟨*proof*⟩

lemma *finite-llast-deadend* [*simp*]: $lfinite\ P \Longrightarrow deadend\ (llast\ P)$
⟨*proof*⟩
lemma *finite-llast-V* [*simp*]: $lfinite\ P \Longrightarrow llast\ P \in V$
⟨*proof*⟩

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*⟩

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**
$strategy\ p\ \sigma \equiv \forall v \in VV\ p.\ \neg deadend\ v \longrightarrow v {\rightarrow} \sigma\ v$

lemma (**in** *ParityGame*) *strategyI* [*intro*]:
$(\bigwedge v.\ [\![\ v \in VV\ p; \neg deadend\ v\ ]\!] \Longrightarrow v {\rightarrow} \sigma\ v) \Longrightarrow strategy\ p\ \sigma$
⟨*proof*⟩

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* σ
| *path-conforms-LCons-LNil*: *path-conforms-with-strategy* p (*LCons* v *LNil*) σ

| *path-conforms-VVp*: $\llbracket$ $v \in VV\ p$; $w = \sigma\ v$; *path-conforms-with-strategy p* $(LCons\ w\ Ps)\ \sigma$ $\rrbracket$
  $\Longrightarrow$ *path-conforms-with-strategy p* $(LCons\ v\ (LCons\ w\ Ps))\ \sigma$
| *path-conforms-VVpstar*: $\llbracket$ $v \notin VV\ p$; *path-conforms-with-strategy p Ps* σ $\rrbracket$
  $\Longrightarrow$ *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-conforms* [*simp*]: *path-conforms-with-strategy p P* σ

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

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

4.3 An Arbitrary Strategy

context *ParityGame* **begin**

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

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

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

4.4 Valid Strategies

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

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

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

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

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

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

lemma *valid-strategy-in-V*: $\llbracket$ *strategy p* σ; $v \in VV\ p$; $\neg$*deadend v* $\rrbracket \Longrightarrow \sigma\ v \in V$
⟨*proof*⟩

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

4.5 Conforming Strategies

lemma *path-conforms-with-strategy-ltl* [*intro*]:
path-conforms-with-strategy p P $\sigma \Longrightarrow$ *path-conforms-with-strategy p* (*ltl P*) σ
⟨*proof*⟩

lemma *path-conforms-with-strategy-drop*:
path-conforms-with-strategy p P $\sigma \Longrightarrow$ *path-conforms-with-strategy p* (*ldropn n P*) σ
⟨*proof*⟩

lemma *path-conforms-with-strategy-prefix*:
path-conforms-with-strategy p P $\sigma \Longrightarrow$ *lprefix* P' $P \Longrightarrow$ *path-conforms-with-strategy p* P' σ
⟨*proof*⟩

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

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

lemma *path-conforms-with-strategy-irrelevant-updates*:
assumes *path-conforms-with-strategy p P* σ $\bigwedge v.\ v \in$ *lset P* $\Longrightarrow \sigma\ v = \sigma'\ v$
shows *path-conforms-with-strategy p P* σ'
⟨*proof*⟩

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

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

lemma *path-conforms-with-strategy-start*:
path-conforms-with-strategy p (*LCons v* (*LCons w P*)) $\sigma \Longrightarrow v \in VV\ p \Longrightarrow \sigma\ v = w$
⟨*proof*⟩

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* $P \in VV\ p \Longrightarrow \sigma$ (*llast* P) $=$ *lhd* P'
shows *path-conforms-with-strategy* p (*lappend* P P') σ
⟨*proof*⟩

lemma *path-conforms-with-strategy-VVpstar*:
assumes *lset* $P \subseteq VV\ p{*}{*}$
shows *path-conforms-with-strategy* p P σ
⟨*proof*⟩

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

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 σ
⟨*proof*⟩

4.6 Greedy Conforming Path

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

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

context begin

private primcorec *greedy-conforming-path* :: *Player* $\Rightarrow$ *'a Strategy* $\Rightarrow$ *'a Strategy* $\Rightarrow$ *'a* $\Rightarrow$ *'a Path*
where
greedy-conforming-path p σ σ' *v0* $=$
LCons v0 (*if deadend v0*
then LNil
else if v0 $\in VV\ p$
then greedy-conforming-path p σ σ' (σ *v0*)
else greedy-conforming-path p σ σ' (σ' *v0*))

private lemma *greedy-path-LNil*: *greedy-conforming-path* p σ σ' *v0* $\neq$ *LNil*
⟨*proof*⟩ **lemma** *greedy-path-lhd*: *greedy-conforming-path* p σ σ' *v0* $=$ *LCons* v $P \Longrightarrow v =$ *v0*
⟨*proof*⟩ **lemma** *greedy-path-deadend-v0*: *greedy-conforming-path* p σ σ' *v0* $=$ *LCons* v $P \Longrightarrow P =$ *LNil* $\longleftrightarrow$ *deadend v0*
⟨*proof*⟩ **corollary** *greedy-path-deadend-v*:
greedy-conforming-path p σ σ' *v0* $=$ *LCons* v $P \Longrightarrow P =$ *LNil* $\longleftrightarrow$ *deadend* v
⟨*proof*⟩
corollary *greedy-path-deadend-v'*: *greedy-conforming-path* p σ σ' *v0* $=$ *LCons* v *LNil* $\Longrightarrow$ *deadend* v

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

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

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

end

end

4.7 Valid Maximal Conforming Paths

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

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

⟨*proof*⟩

lemma *extension-valid-maximal-conforming*:
assumes $v' \rightarrow v0$ $v' \in VV\ p \Longrightarrow \sigma\ v' = v0$
shows *vmc-path G* (*LCons* v' *P*) v' *p* σ
⟨*proof*⟩

lemma *vmc-path-ldropn*:
assumes *enat n* < *llength P*
shows *vmc-path G* (*ldropn n P*) (*P* \$ *n*) *p* σ
⟨*proof*⟩

lemma *conforms-to-another-strategy*:
path-conforms-with-strategy p P $\sigma' \Longrightarrow$ *vmc-path G P v0 p* σ'
⟨*proof*⟩
end

lemma (**in** *ParityGame*) *valid-maximal-conforming-path-0*:
assumes ¬*lnull P valid-path P maximal-path P path-conforms-with-strategy p P* σ
shows *vmc-path G P* (*P* \$ *0*) *p* σ
⟨*proof*⟩

4.8 Valid Maximal Conforming Paths with One Edge

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

locale *vmc-path-no-deadend* = *vmc-path* +
assumes *v0-no-deadend* [*simp*]: ¬*deadend v0*
begin
definition *w0* ≡ *lhd* (*ltl P*)

lemma *Ptl-not-null* [*simp*]: ¬*lnull* (*ltl P*)
⟨*proof*⟩
lemma *Ptl-LCons*: *ltl P* = *LCons w0* (*ltl* (*ltl P*)) ⟨*proof*⟩
lemma *P-LCons'*: *P* = *LCons v0* (*LCons w0* (*ltl* (*ltl P*))) ⟨*proof*⟩
lemma *v0-edge-w0* [*simp*]: $v0 \rightarrow w0$ ⟨*proof*⟩

lemma *Ptl-0*: *ltl P* \$ *0* = *lhd* (*ltl P*) ⟨*proof*⟩
lemma *P-Suc-0*: *P* \$ *Suc 0* = *w0* ⟨*proof*⟩
lemma *Ptl-edge* [*simp*]: *v0* → *lhd* (*ltl P*) ⟨*proof*⟩

lemma *v0-conforms*: $v0 \in VV\ p \Longrightarrow \sigma\ v0 = w0$
⟨*proof*⟩

lemma *w0-V* [*simp*]: $w0 \in V$ ⟨*proof*⟩
lemma *w0-lset-P* [*simp*]: $w0 \in$ *lset P* ⟨*proof*⟩

lemma *vmc-path-ltl* [*simp*]: *vmc-path G* (*ltl P*) *w0 p* σ ⟨*proof*⟩
end

context *vmc-path* **begin**

lemma *vmc-path-lnull-ltl-no-deadend*:
  $\neg$*lnull* (*ltl P*) $\Longrightarrow$ *vmc-path-no-deadend G P v0 p* σ
  ⟨*proof*⟩

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

4.9 *lset* Induction Schemas for Paths

Let us define an induction schema useful for proving *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*. ⟦ *vmc-path-no-deadend G P v0 p* σ; *v0* $\in$ *S*; *Q P* ⟧
      $\Longrightarrow$ *Q* (*ltl P*) $\wedge$ (*vmc-path-no-deadend.w0 P*) $\in$ *S*
  shows *lset P* $\subseteq$ *S*
⟨*proof*⟩

⟦*?Q P*; *v0* $\in$ *?S*; $\bigwedge$*P v0*. ⟦*vmc-path-no-deadend G P v0 p* σ; *v0* $\in$ *?S*; *?Q P*⟧ $\Longrightarrow$ *?Q* (*ltl P*) $\wedge$ *vmc-path-no-deadend.w0 P* $\in$ *?S*⟧ $\Longrightarrow$ *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*. ⟦ *vmc-path-no-deadend G P v0 p* σ; *v0* $\in$ *S* ⟧
      $\Longrightarrow$ *vmc-path-no-deadend.w0 P* $\in$ *S*
  shows *lset P* $\subseteq$ *S*
  ⟨*proof*⟩

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*. ⟦ *v* $\in$ *S*; $\neg$*deadend v*; *v* $\in$ *VV p* ⟧ $\Longrightarrow$ σ *v* $\in$ *S* $\cup$ *T*
    and *VVpstar*: $\bigwedge$*v w*. ⟦ *v* $\in$ *S*; $\neg$*deadend v*; *v* $\in$ *VV p*** ; *v*$\rightarrow$*w* ⟧ $\Longrightarrow$ *w* $\in$ *S* $\cup$ *T*
    and *v0*: *v0* $\in$ *S*
    and *disjoint*: *lset P* $\cap$ *T* $=$ {}
  shows *lset P* $\subseteq$ *S*
⟨*proof*⟩

end

end

5 Attracting Strategies

theory *AttractingStrategy*
imports
  Main
  Strategy

begin

Here we introduce the concept of attracting strategies.

context *ParityGame* **begin**

5.1 Paths Visiting a Set

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

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

lemma *visits-via-monotone*: $[\![\ \textit{visits-via } P\ A\ W;\ A \subseteq A'\ ]\!] \Longrightarrow \textit{visits-via } P\ A'\ W$
  ⟨*proof*⟩

lemma *visits-via-visits*: $\textit{visits-via } P\ A\ W \Longrightarrow \textit{lset } P \cap W \neq \{\}$
  ⟨*proof*⟩

lemma (**in** *vmc-path*) *visits-via-trivial*: $v0 \in W \Longrightarrow \textit{visits-via } P\ A\ W$
  ⟨*proof*⟩

lemma *visits-via-LCons*:
  assumes *visits-via P A W*
  shows *visits-via* (*LCons v0 P*) (*insert v0 A*) *W*
⟨*proof*⟩

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

lemma (**in** *vm-path*) *visits-via-deadend*:
  assumes *visits-via P A* (*deadends p*)
  shows *winning-path p*** *P*
  ⟨*proof*⟩

5.2 Attracting Strategy from a Single Node

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

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

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

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

$\langle proof \rangle$

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

lemma *strategy-attracts-via-v0*:
assumes σ: *strategy p* σ *strategy-attracts-via p* σ *v0 A W*
and *v0*: *v0* $\in$ *V*
shows *v0* $\in$ *A* $\cup$ *W*
$\langle proof \rangle$
corollary *strategy-attracts-not-outside*:
$[\![$ *v0* $\in$ *V* $-$ *A* $-$ *W*; *strategy p* σ $]\!] \Longrightarrow \neg$*strategy-attracts-via p* σ *v0 A W*
$\langle proof \rangle$

lemma *strategy-attracts-viaI* [*intro*]:
assumes $\bigwedge P$. *vmc-path G P v0 p* $\sigma \Longrightarrow$ *visits-via P A W*
shows *strategy-attracts-via p* σ *v0 A W*
$\langle proof \rangle$

lemma *strategy-attracts-via-no-deadends*:
assumes *v* $\in$ *V v* $\in$ *A* $-$ *W strategy-attracts-via p* σ *v A W*
shows $\neg$*deadend v*
$\langle proof \rangle$

lemma *attractor-strategy-on-extends*:
$[\![$ *strategy-attracts-via p* σ *v0 A W*; *A* $\subseteq$ *A'* $]\!] \Longrightarrow$ *strategy-attracts-via p* σ *v0 A' W*
$\langle proof \rangle$

lemma *strategy-attracts-via-trivial*: *v0* $\in$ *W* $\Longrightarrow$ *strategy-attracts-via p* σ *v0 A W*
$\langle proof \rangle$

lemma *strategy-attracts-via-successor*:
assumes σ: *strategy p* σ *strategy-attracts-via p* σ *v0 A W*
and *v0*: *v0* $\in$ *A* $-$ *W*
and *w0*: *v0*$\rightarrow$*w0 v0* $\in$ *VV p* $\Longrightarrow \sigma$ *v0* = *w0*
shows *strategy-attracts-via p* σ *w0 A W*
$\langle proof \rangle$

lemma *strategy-attracts-VVp*:
assumes σ: *strategy p* σ *strategy-attracts-via p* σ *v0 A W*
and *v*: *v0* $\in$ *A* $-$ *W v0* $\in$ *VV p* $\neg$*deadend v0*
shows σ *v0* $\in$ *A* $\cup$ *W*
$\langle proof \rangle$

lemma *strategy-attracts-VVpstar*:
assumes *strategy p* σ *strategy-attracts-via p* σ *v0 A W*
and *v0* $\in$ *A* $-$ *W v0* $\notin$ *VV p w0* $\in$ *V* $-$ *A* $-$ *W*
shows $\neg$*v0* $\rightarrow$ *w0*
$\langle proof \rangle$

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 σ *A W* $\equiv \forall$ *v0* $\in$ *A. strategy-attracts-via p* σ *v0 A W*

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

lemma *strategy-attractsI* [*intro*]:
assumes $\bigwedge$*P v*. ⟦ *v* $\in$ *A*; *vmc-path G P v p* σ ⟧ $\Longrightarrow$ *visits-via P A W*
shows *strategy-attracts p* σ *A W*
⟨*proof*⟩

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

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

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

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

lemma *strategy-attracts-does-not-leave*:
assumes σ: *strategy-attracts p* σ *A W strategy p* σ
and *v*: *v*$\rightarrow$*w v* $\in$ *A* $-$ *W w* $\notin$ *A* $\cup$ *W*
shows *v* $\in$ *VV p* $\wedge$ σ *v* $\neq$ *w*
⟨*proof*⟩

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

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

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

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

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

and σ: *strategy-attracts* $p\ \sigma\ A\ W$
and *P-hits-A*: *lset* $P \cap A \neq \{\}$
shows *lset* $P \cap W \neq \{\}$
⟨*proof*⟩

lemma *attracted-strategy-step*:
assumes σ: *strategy* $p\ \sigma$ *strategy-attracts* $p\ \sigma\ A\ W$
and *v0*: $\neg$*deadend v0 v0* $\in A - W$ *v0* $\in VV\ p$
shows σ *v0* $\in A \cup W$
⟨*proof*⟩

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* $\in A \cup W$
⟨*proof*⟩

end — context ParityGame

end

6 Attractor Sets

theory *Attractor*
imports
Main
AttractingStrategy
begin

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

context *ParityGame* **begin**

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

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

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

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

definition *attractor* :: *Player* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* **where**
attractor $p\ W$ = *lfp* (*attractor-step* $p\ W$)

6.1 *directly-attracted*

Show a few basic properties of *directly-attracted*.

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

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

6.2 *attractor-step*

lemma *attractor-step-empty*: *attractor-step* p $\{\}$ $\{\} = \{\}$
and *attractor-step-bounded-by-V*: $[\![$ $W \subseteq V$; $S \subseteq V$ $]\!] \Longrightarrow$ *attractor-step* p W $S \subseteq V$
⟨*proof*⟩

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

lemma *attractor-step-mono*: *mono* (*attractor-step* p W)
⟨*proof*⟩

6.3 Basic Properties of an Attractor

lemma *attractor-unfolding*: *attractor* p W = *attractor-step* p W (*attractor* p W)
⟨*proof*⟩
lemma *attractor-lowerbound*: *attractor-step* p W $S \subseteq S \Longrightarrow$ *attractor* p $W \subseteq S$
⟨*proof*⟩
lemma *attractor-set-non-empty*: $W \neq \{\} \Longrightarrow$ *attractor* p $W \neq \{\}$
and *attractor-set-base*: $W \subseteq$ *attractor* p W
⟨*proof*⟩
lemma *attractor-in-V*: $W \subseteq V \Longrightarrow$ *attractor* p $W \subseteq V$
⟨*proof*⟩

6.4 Attractor Set Extensions

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

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

6.5 Removing an Attractor

lemma *removing-attractor-induces-no-deadends*:
assumes $v \in S -$ *attractor* p W $v{\rightarrow}w$ $w \in S$ $\bigwedge w.$ $[\![$ $v \in VV\ p{**}$; $v{\rightarrow}w$ $]\!] \Longrightarrow w \in S$
shows $\exists\, w \in S -$ *attractor* p $W.$ $v{\rightarrow}w$
⟨*proof*⟩

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

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

shows $\neg$*Digraph.deadend* (*subgame* V') v
⟨*proof*⟩

6.6 Attractor Set Induction

lemma *mono-restriction-is-mono*: *mono* $f \Longrightarrow$ *mono* $(\lambda S.\ f\ (S \cap V))$
⟨*proof*⟩

Here we prove a powerful induction schema for *attractor*. Being able to prove this is the only reason why we do not use `inductive_set` to define the attractor set.

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

lemma *attractor-set-induction* [*consumes 1*, *case-names step union*]:
assumes $W \subseteq V$
and *step*: $\bigwedge S.\ S \subseteq V \Longrightarrow P\ S \Longrightarrow P$ (*attractor-step* $p\ W\ S$)
and *union*: $\bigwedge M.\ \forall S \in M.\ S \subseteq V \wedge P\ S \Longrightarrow P\ (\bigcup M)$
shows P (*attractor* $p\ W$)
⟨*proof*⟩

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

lemma *winning-strategyI* [*intro*]:
assumes $\bigwedge P.$ *vmc-path* $G\ P$ *v0* $p\ \sigma \Longrightarrow$ *winning-path* $p\ P$
shows *winning-strategy* $p\ \sigma$ *v0*
⟨*proof*⟩

lemma (**in** *vmc-path*) *paths-hits-winning-strategy-is-winning*:
assumes σ: *winning-strategy* $p\ \sigma\ v$
and v: $v \in$ *lset* P
shows *winning-path* $p\ P$
⟨*proof*⟩

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

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

assumes σ: *strategy p* σ *winning-strategy p* σ *v*
and σ': *strategy p*∗∗ σ' *winning-strategy p*∗∗ σ' *v*
and *v*: $v \in V$
shows *False*
⟨*proof*⟩

7.1 Deadends

lemma *no-winning-strategy-on-deadends*:
assumes $v \in VV\ p$ *deadend v strategy p* σ
shows ¬*winning-strategy p* σ *v*
⟨*proof*⟩

lemma *winning-strategy-on-deadends*:
assumes $v \in VV\ p$ *deadend v strategy p* σ
shows *winning-strategy p*∗∗ σ *v*
⟨*proof*⟩

7.2 Extension Theorems

lemma *strategy-extends-VVp*:
assumes *v0*: $v0 \in VV\ p$ ¬*deadend v0*
and σ: *strategy p* σ *winning-strategy p* σ *v0*
shows *winning-strategy p* σ (σ *v0*)
⟨*proof*⟩

lemma *strategy-extends-VVpstar*:
assumes *v0*: $v0 \in VV\ p{**}\ v0 \rightarrow w0$
and σ: *winning-strategy p* σ *v0*
shows *winning-strategy p* σ *w0*
⟨*proof*⟩

lemma *strategy-extends-backwards-VVpstar*:
assumes *v0*: $v0 \in VV\ p{**}$
and σ: *strategy p* σ $\bigwedge w.\ v0 \rightarrow w \Longrightarrow$ *winning-strategy p* σ *w*
shows *winning-strategy p* σ *v0*
⟨*proof*⟩

lemma *strategy-extends-backwards-VVp*:
assumes *v0*: $v0 \in VV\ p\ \sigma\ v0 = w\ v0 \rightarrow w$
and σ: *strategy p* σ *winning-strategy p* σ *w*
shows *winning-strategy p* σ *v0*
⟨*proof*⟩

end — context ParityGame

end

8 Well-Ordered Strategy

theory *WellOrderedStrategy*

imports
Main
Strategy
begin

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

The following locale formalizes this idea.

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

locale *WellOrderedStrategies* = *ParityGame* +
fixes S :: $'a$ *set*
and p :: *Player*
— The set of good strategies on a node v
and *good* :: $'a \Rightarrow 'a$ *Strategy set*
and r :: ($'a$ *Strategy* × $'a$ *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* $\{\sigma.\ \exists\, v \in S.\ \sigma \in good\ v\}\ r$
— Every node has a good strategy.
and *good-ex*: $\bigwedge v.\ v \in S \Longrightarrow \exists\, \sigma.\ \sigma \in good\ v$
— good strategies are well-formed strategies.
and *good-strategies*: $\bigwedge v\ \sigma.\ \sigma \in good\ v \Longrightarrow strategy\ p\ \sigma$
— A good strategy on v is also good on possible successors of v.
and *strategies-continue*: $\bigwedge v\ w\ \sigma.\ [\![\ v \in S;\ v{\rightarrow}w;\ v \in VV\ p \Longrightarrow \sigma\ v = w;\ \sigma \in good\ v\ ]\!] \Longrightarrow \sigma \in good\ w$
begin

The set of all strategies which are good somewhere.

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

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

no-notation *binomial* (**infixl** *choose 65*)

Among the good strategies on v, choose the minimum.

definition *choose* **where**
choose $v \equiv$ *THE* σ. *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 $\equiv$ *override-on* σ*-arbitrary* ($\lambda v.$ *choose* $v\ v$) S

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

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

lemma *r-total* [*simp*]: *total-on Strategies r*
⟨*proof*⟩
lemma *r-trans* [*simp*]: *trans r*
⟨*proof*⟩
lemma *r-wf* [*simp*]: *wf* (*r* − *Id*)
⟨*proof*⟩

choose always chooses a minimal good strategy on *S*.

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

corollary
assumes $v \in S$
shows *choose-good*: *choose v* ∈ *good v*
and *choose-minimal*: $\bigwedge\sigma'$. (σ', *choose v*) ∈ *r* − *Id* ⟹ $\sigma' \notin$ *good v*
and *choose-strategy*: *strategy p* (*choose v*)
⟨*proof*⟩

corollary *choose-in-Strategies*: $v \in S$ ⟹ *choose v* ∈ *Strategies* ⟨*proof*⟩

lemma *well-ordered-strategy-valid*: *strategy p well-ordered-strategy*
⟨*proof*⟩

8.1 Strategies on a Path

Maps a path to its strategies.

definition *path-strategies* ≡ *lmap choose*

lemma *path-strategies-in-Strategies*:
assumes *lset P* ⊆ *S*
shows *lset* (*path-strategies P*) ⊆ *Strategies*
⟨*proof*⟩

lemma *path-strategies-good*:
assumes *lset P* ⊆ *S enat n* < *llength P*
shows *path-strategies P* \$ *n* ∈ *good* (*P* \$ *n*)
⟨*proof*⟩

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

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

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

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

8.2 Eventually One Strategy

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

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

end — WellOrderedStrategies

end

9 Winning Regions

theory *WinningRegion*
imports
Main
WinningStrategy
begin

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

context *ParityGame* **begin**

definition *winning-region p* $\equiv$ { *v* $\in$ *V*. $\exists$ *σ*. *strategy p σ* $\wedge$ *winning-strategy p σ v* }

lemma *winning-regionI* [*intro*]:
assumes *v* $\in$ *V strategy p σ winning-strategy p σ v*
shows *v* $\in$ *winning-region p*
⟨*proof*⟩

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

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

⟨*proof*⟩

9.1 Paths in Winning Regions

lemma (**in** *vmc-path*) *paths-stay-in-winning-region*:
  assumes σ': *strategy p* σ' *winning-strategy p* σ' *v0*
    and σ: $\bigwedge v.\ v \in$ *winning-region p* $\Longrightarrow \sigma'\ v = \sigma\ v$
  shows *lset P* $\subseteq$ *winning-region p*
⟨*proof*⟩

lemma (**in** *vmc-path*) *path-hits-winning-region-is-winning*:
  assumes σ': *strategy p* σ' $\bigwedge v.\ v \in$ *winning-region p* $\Longrightarrow$ *winning-strategy p* $\sigma'\ v$
    and σ: $\bigwedge v.\ v \in$ *winning-region p* $\Longrightarrow \sigma'\ v = \sigma\ v$
    and P: *lset P* $\cap$ *winning-region p* $\neq \{\}$
  shows *winning-path p P*
⟨*proof*⟩

9.2 Irrelevant Updates

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

lemma *winning-strategy-updates*:
  assumes σ: *strategy p* σ *winning-strategy p* σ *v0*
    and v: $v \notin$ *winning-region p* $v \rightarrow w$
  shows *winning-strategy p* $(\sigma(v := w))$ *v0*
⟨*proof*⟩

9.3 Extending Winning Regions

lemma *winning-region-extends-VVp*:
  assumes v: $v \in$ *VV p* $v \rightarrow w$ **and** w: $w \in$ *winning-region p*
  shows $v \in$ *winning-region p*
⟨*proof*⟩

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

end — context ParityGame

end

10 Uniform Strategies

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

theory *UniformStrategy*
imports
  Main
  AttractingStrategy WinningStrategy WellOrderedStrategy WinningRegion
begin

context *ParityGame* **begin**

10.1 A Uniform Attractor Strategy

lemma *merge-attractor-strategies*:
  assumes $S \subseteq V$
    and *strategies-ex*: $\bigwedge v.\ v \in S \Longrightarrow \exists \sigma.\ \mathit{strategy}\ p\ \sigma \wedge \mathit{strategy\text{-}attracts\text{-}via}\ p\ \sigma\ v\ S\ W$
  shows $\exists \sigma.\ \mathit{strategy}\ p\ \sigma \wedge \mathit{strategy\text{-}attracts}\ p\ \sigma\ S\ W$
⟨*proof*⟩

10.2 A Uniform Winning Strategy

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

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

10.3 Extending Winning Regions

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

lemma *winning-region-extends-VVpstar*:
  assumes v: $v \in VV\ p{*}{*}$ **and** w: $\bigwedge w.\ v{\rightarrow}w \Longrightarrow w \in \mathit{winning\text{-}region}\ p$
  shows $v \in \mathit{winning\text{-}region}\ p$
⟨*proof*⟩

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

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

end — context ParityGame

end

11 Attractor Strategies

theory *AttractorStrategy*
imports
  Main
  Attractor UniformStrategy
begin

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

context *ParityGame* **begin**

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

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

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

11.1 Existence

Prove that every attractor set has an attractor strategy.

theorem *attractor-has-strategy*:
  assumes $W \subseteq V$
  shows $\exists\sigma$. *strategy p* σ $\wedge$ *strategy-attracts p* σ (*attractor p W*) *W*
⟨*proof*⟩

end — context ParityGame

end

12 Positional Determinacy of Parity Games

theory *PositionalDeterminacy*
imports
  Main
  AttractorStrategy
begin

context *ParityGame* **begin**

12.1 Induction Step

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

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

lemma *positional-strategy-induction-step*:
  assumes $v \in V$
    and *no-deadends*: $\bigwedge v.\ v \in V \Longrightarrow \neg$*deadend v*
    and *IH*: $\bigwedge$(*G* :: ('*a*, '*b*) *ParityGame-scheme*) *v*.
      ⟦ *card* (ω_G ‘ V_G) < *card* (ω ‘ V); $v \in V_G$;
        ParityGame G;
        $\bigwedge v.\ v \in V_G \Longrightarrow \neg$*Digraph.deadend G v* ⟧
      $\Longrightarrow \exists p.\ v \in$ *ParityGame.winning-region G p*

shows $\exists p.\ v \in$ *winning-region p*
⟨*proof*⟩

12.2 Positional Determinacy without Deadends

theorem *positional-strategy-exists-without-deadends*:
assumes $v \in V$ $\bigwedge v.\ v \in V \Longrightarrow \neg$*deadend v*
shows $\exists p.\ v \in$ *winning-region p*
⟨*proof*⟩

12.3 Positional Determinacy with Deadends

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

theorem *positional-strategy-exists*:
assumes $v0 \in V$
shows $\exists p.\ v0 \in$ *winning-region p*
⟨*proof*⟩

12.4 The Main Theorem: Positional Determinacy

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

theorem *partition-into-winning-regions*:
shows $V =$ *winning-region Even* $\cup$ *winning-region Odd*
and *winning-region Even* $\cap$ *winning-region Odd* $= \{\}$
⟨*proof*⟩

end — context ParityGame

end

13 Defining the Attractor with `inductive_set`

theory *AttractorInductive*
imports
Main
Attractor
begin

context *ParityGame* **begin**

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

13.1 *attractor-inductive*

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

inductive-set *attractor-inductive* :: *Player* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set*
for *p* :: *Player* **and** *W* :: *'a set* **where**
Base [*intro*!]: $v \in W \Longrightarrow v \in$ *attractor-inductive p W*
| *VVp*: $[\![\ v \in VV\ p;\ \exists\, w.\ v{\to}w \land w \in$ *attractor-inductive p W* $]\!]$
$\Longrightarrow v \in$ *attractor-inductive p W*
| *VVpstar*: $[\![\ v \in VV\ p{**};\ \neg deadend\ v;\ \forall\, w.\ v{\to}w \longrightarrow w \in$ *attractor-inductive p W* $]\!]$
$\Longrightarrow v \in$ *attractor-inductive p W*

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

lemma *attractor-inductive-is-attractor*:
assumes $W \subseteq V$
shows *attractor-inductive p W* = *attractor p W*
$\langle proof \rangle$

end

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://afp.sf.net/entries/Coinductive.shtml, Formal proof development.

[5] Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.