Analysing the complexity of games on graphs

MSc Thesis (*Afstudeerscriptie*)

written by

Samson Tikitu de Jager
(born May 30th, 1981 in Takaka, New Zealand)

under the supervision of **Dr Benedikt Löwe**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam*.

Date of the public defense:	**Members of the Thesis Committee:**
August 29, 2005	Prof.dr Peter van Emde Boas (chair)
	Prof.dr Johan van Benthem
	Dr Benedikt Löwe

Institute for Logic, Language and Computation

Contents

Chapter 0

Introduction and definitions

This thesis does not have a thesis, in the sense of a central statement that is developed and argued for. Instead, it explores three different ways of describing complexity in simple games.

The three approaches are drawn from set theory, from automata theory, and from theory of algorithms and computer science. These three areas, while obviously related, generally use different notation and different approaches even while ostensibly talking about the same things: 'games'.

The progression of ideas through this thesis could be seen as a movement from the abstract to the concrete. The formalism presented in this chapter is the descriptive set theory notation, an explicitly infinitary concept and one that, for sufficiently complicated games, can be shown not to yield to constructive methods of proof (by comparison, however, the games considered in this thesis have been chosen specifically because such methods *are* successful). Chapter 1 presents the topological notion of complexity from descriptive set theory, and some abstract solution concepts utilising infinitary procedures on infinitary objects.

In Chapter 2 we move to automata theory, restricting ourselves to a class of games that are played on finite structures (the games themselves remain infinite, but can be given finite representations). The focus here is on the finite computational resources needed to implement a strategy, which in the descriptive set theoretic context might easily be infinite.

Chapter 3 takes an algorithmic approach to the process of *constructing* a strategy for an automaton game, and measures the complexity of the game by the time complexity of the algorithm to solve it. This is the most concrete of the three approaches, in fact a Java application implementing some of these algorithms accompanies the written text of this thesis. Not only are the games restricted to finite representations (automata games, on finite structures) and the strategies implementable with finite computational resources, but the process of producing a strategy is also finitary and algorithmic.

The final chapter concerns a software implementation of some of the algorithms given in Chapter 3, consisting of a Java library and a graphical applica-

tion, for experimenting with graph games and labellings.

A short Appendix extends the techniques of Chapters 1, 2 and 3 to an additional hierarchy of games, giving a finer-grained analysis of one of the classes discussed in Chapter 1.

0.1 Preliminaries

The class of ordinals is denoted by Ord, the natural numbers by ω.

Definition 0.1 (Sequences). A SEQUENCE of elements from a set X is a total function $s\colon \alpha \to X$ where $\alpha \leq \omega$ is a natural number or ω, the LENGTH of s (denoted $\mathrm{lh}(s)$). We write sequences in angle brackets $\langle 1, 2, 3\rangle$, and $\langle\rangle$ denotes the empty sequence. By analogy with set notation, we also write $a = \langle a_i \,;\, i < n\rangle$ for a sequence a of length n such that $a(i) = a_i$ for $i < n$.

A sequence s is FINITE if $\mathrm{lh}(s) < \omega$, otherwise INFINITE. The set of finite sequences of elements from a set X is denoted by ${}^{<\omega}X$, the infinite sequences by ${}^{\omega}X$, and ${}^{\leq\omega}X \stackrel{\text{def}}{=} {}^{<\omega}X \cup {}^{\omega}X$.

Some ordinary function notions are also useful for sequences. The length of a sequence s is also its DOMAIN $\mathrm{dom}(s)$. (The RANGE of s, denoted $\mathrm{ran}(s)$, collects the elements appearing in s.) The RESTRICTION of s

$$s{\restriction}n \stackrel{\text{def}}{=} \langle s(i) \,;\, i < n\rangle$$

corresponds to the intuitive notion of 'cutting off' the sequence at (just before) the index n. Given two sequences s and t (over some set X), we say s is an INITIAL SUBSEQUENCE of t if $s \subseteq t$ (and a PROPER INITIAL SUBSEQUENCE if $s \subsetneq t$, also that t EXTENDS s).

For s a finite sequence and t finite or infinite, we define the CONCATENATION $s^\frown t$ as follows:

$$(s^\frown t)(i) \stackrel{\text{def}}{=} \begin{cases} s(i) & \text{if } i < \mathrm{lh}(s), \\ t(i - \mathrm{lh}(s)) & \text{otherwise.} \end{cases}$$

For $x \in X$, we often write $s^\frown x$ for the SINGLETON EXTENSION $s^\frown\langle x\rangle$. We write $\langle x^n\rangle$ for the constant sequence $i \mapsto x$ of length n, $\langle x^n y^m\rangle$ for $\langle x^n\rangle^\frown\langle y^m\rangle$, etc.

Definition 0.2 (Graphs). A GRAPH is a pair $G = \langle V, E\rangle$ where V is an arbitrary VERTEX SET and $E \subseteq V \times V$ the EDGE SET or EDGE RELATION.

The SUCCESSOR and PREDECESSOR functions from V to $\wp(V)$ are defined by

$$\mathrm{succ}_G(v) \stackrel{\text{def}}{=} \{v' \in V \,;\, \langle v, v'\rangle \in E\},$$
$$\mathrm{pred}_G(v) \stackrel{\text{def}}{=} \{v' \in V \,;\, \langle v', v\rangle \in E\}.$$

(We omit the subscript wherever natural.) A vertex v such that $\mathrm{succ}_G(v) = \varnothing$ is known as a LEAF. A graph without leaves is PRUNED.

A WALK through a graph $\langle V, E\rangle$ is a sequence $s \in {}^{\leq\omega}V$ such that for all $n < \mathrm{lh}(s) - 1$, $s(n)\ E\ s(n+1)$. If all vertices in a walk s are distinct, then

s is called a PATH. A CYCLE is a walk with at least one repeated vertex (a walk s such that for $n < m < \mathrm{lh}(s)$, $s(n) = s(m)$). Given a graph G the sets walks(G) and paths(G) collect respectively the walks and paths (including the empty walk $\langle\rangle$).

Definition 0.3 (Trees). A TREE on an arbitrary set X is a set $T \subseteq {}^{<\omega}X$ closed under initial subsequence. (That is, if $s, t \in {}^{<\omega}X$, $s \in T$ and $t \subseteq s$, then $t \in T$.) We can view a tree T as a graph $\langle T, E\rangle$ with E given by

$$\langle s, t\rangle \in T \overset{\text{def}}{\iff} \exists x{\in}X(t = s^\frown x).$$

The tree definitions of leaves, a pruned tree, and the successor and predecessor functions lift in the obvious manner.

Given a tree $T \subseteq {}^{<\omega}X$, $[T] \subseteq {}^{\omega}X$ denotes the set of BRANCHES through T:

$$[T] \overset{\text{def}}{=} \{a \in {}^{\omega}X \,;\, \forall n{\in}\omega(a{\restriction}n \in T)\}.$$

The SUBTREE INDUCED BY a sequence $s \in T$, denoted T_s, is defined by

$$T_s \overset{\text{def}}{=} \{t \in T \,;\, s \subseteq t \vee t \subseteq s\}.$$

A related notion is the BRANCHES INDUCED BY $s \in T$, denoted $[s]_T$ and defined by

$$[s]_T \overset{\text{def}}{=} \{a \in [T] \,;\, s \subseteq a\} = [T_s].$$

(We omit the subscript if this is clear from context.)

Definition 0.4 (Unravelling a graph). If $G = \langle V, E\rangle$ is a graph, and $v_0 \in V$ an arbitrary vertex, then we can define a tree $T \subseteq {}^{<\omega}V$ called the UNRAVELLED TREE OF G with root v_0:

$$T \overset{\text{def}}{=} \{s \in \mathrm{walks}(G) \,;\, s = \langle\rangle \vee s(0) \in \mathrm{succ}_G(v_0)\}.$$

0.2 Games

The games considered in this thesis are infinite alternating two-player games of perfect information, played on a countable move set. Intuitively, the players **I** and **II** take turns picking a move from a set of possibilities. The resulting sequence after ω moves is used to decide the outcome of the game.

Definition 0.5 (Games). A GAME G with countable MOVE SET X is a pair $\langle T, \Omega\rangle$, where $T \subseteq {}^{<\omega}X$ is a non-empty pruned tree, and $\Omega\colon [T] \to \{\mathbf{I}, \mathbf{D}, \mathbf{II}\}$ is the PAYOFF FUNCTION.

The set $[T]$ is the possible PLAYS of G, and Ω assigns an OUTCOME to each play. The three outcomes represent respectively a WIN for **I** (and LOSS for **II**), a DRAW, and a win for **II** (loss for **I**). We use shorthand notation $\Omega_{\mathbf{I}}$ for $\{a \in [T] \,;\, \Omega(a) = \mathbf{I}\}$, and so on.

A sequence $s \in T$ is called a POSITION, and the set $\{x \in X \,;\, s^\frown x \in T\}$ is the MOVES AVAILABLE FROM s. A position s is OWNED BY **I** if $\mathrm{lh}(s)$ is even, otherwise it is owned by **II**.

For positions $s \in T$, T_s is known as the SUBGAME FROM s. (Note that $[T_s] = [s]_T$.)

Conceptually speaking, to play a game the players alternate choosing elements from X and thus construct a branch through ${}^\omega X$. We use the notion of a strategy both to formalise "choosing an element" and to constrain the resulting branch to one through T.

Definition 0.6 (Strategies). Let $T \subseteq {}^{<\omega}X$ be a pruned tree. A STRATEGY for the game $\langle T, \Omega\rangle$ for player **I** is a pruned tree $\sigma \subseteq T$ such that for all $s \in \sigma$, if $\mathrm{lh}(s)$ is even then $|\mathrm{succ}_\sigma(s)| = 1$, and if $\mathrm{lh}(s)$ is odd then $\mathrm{succ}_\sigma(s) = \mathrm{succ}_T(s)$. A strategy for **II** exchanges the words "odd" and "even" in the definition. The branches through the strategy, $[\sigma]$, are the PLAYS ACCORDING TO σ.

For convenience, we use a function-like shorthand for expressing the 'choices' strategies make: if σ is a strategy for player i and $s \in \sigma$ is a position owned by i, then $\sigma(s)$ denotes the unique $t \in \sigma$ such that $\mathrm{succ}_\sigma(s) = \{t\}$.

Intuitively, if σ is a strategy for player i, then for every position s owned by i, σ chooses one available move (denoted $\sigma(s)$), while σ includes every available move for positions owned by the opponent.

Definition 0.7 (Winning strategies). A strategy σ for **I** in the game $\langle T, \Omega\rangle$ is WINNING if $[\sigma] \subseteq \Omega_{\mathbf{I}}$, and likewise for **II**. Similarly, σ is NON-LOSING (for **I**) if $[\sigma] \subseteq \Omega_{\mathbf{I}} \cup \Omega_{\mathbf{D}}$.

For arbitrary $s \in \sigma$ we say that σ is WINNING FROM s if $[\sigma] \cap [s] \subseteq \Omega_{\mathbf{I}}$, and likewise for **II** and the other outcomes. (Equivalently, if σ is winning in the subgame from s.) A game G is called DETERMINED if one of the players has a winning strategy for G.

Definition 0.8 (Strategy equivalence). Let σ and τ be strategies for the games $\langle T, \Omega\rangle$ and $\langle T', \Omega'\rangle$, respectively. We say they are PLAY EQUIVALENT if $[\sigma] = [\tau]$. We say that σ and τ are OUTCOME EQUIVALENT if $\{\Omega(a) \,;\, a \in [\sigma]\} = \{\Omega(a) \,;\, a \in [\tau]\}$.

Play equivalence is a very strong notion of equivalence. Outcome equivalence, on the other hand, is very weak indeed. For instance, any two winning strategies (for the same player) are outcome equivalent.

Definition 0.9 (Games with rule). Let $G = \langle T, \Omega\rangle$ be an arbitrary game where $T \subsetneq {}^{<\omega}X$. We say G is a game WITH RULE T.

Define $\Omega' \colon {}^\omega X \to \{\mathbf{I}, \mathbf{D}, \mathbf{II}\}$ by

$$\Omega'(a) = \begin{cases} \Omega(a) & \text{if } a \in [T], \\ \mathbf{I} & \text{if } a \notin [T] \text{ and } \max\{n \,;\, a{\restriction}n \in T\} \text{ is odd}, \\ \mathbf{II} & \text{if } a \notin [T] \text{ and } \max\{n \,;\, a{\restriction}n \in T\} \text{ is even}. \end{cases}$$

The game $\langle {}^{<\omega}X, \Omega'\rangle$ is the RULIFICATION of $\langle T, \Omega\rangle$.

Intuitively, the rulification is the original game extended by the 'rule' "the first player to play outside the tree loses." The following theorem justifies the connection between these two games.

Theorem 0.10. *Let $G = \langle T, \Omega \rangle$ be an arbitrary game with rule, and $G' = \langle {}^{<\omega}X, \Omega' \rangle$ its rulification. For any strategy $\sigma \subseteq T$ winning or non-losing in G, there exists a strategy $\sigma' \supseteq \sigma$ for G' that is outcome-equivalent to σ.*

Proof. Only one of the cases will be proved (that of a winning strategy for **I**), the proofs for the other cases are analogous. The idea is that we can build a strategy that always stays within the tree. If a play exits the tree, the strategy guarantees that it will be the opponent's fault.

Let σ be a winning strategy for **I** in G. Define σ' as follows, taking x_0 an arbitrary element of X:

1. If $s \in \sigma$ then $s \in \sigma'$;
2. If $s \in \sigma'$ with $\mathrm{lh}(s)$ odd, then $\mathrm{succ}_{\sigma'}(s) = \mathrm{succ}_{{}^{<\omega}X}(s)$;
3. If $s \in \sigma'$ with $\mathrm{lh}(s)$ even, and $\mathrm{succ}_{\sigma}(s) = \varnothing$, then $s^\frown x_0 \in \sigma'$.

Obviously σ' is a strategy for **I**. Let $a \subseteq [\sigma']$ be an arbitrary path. Either $a \in [\sigma]$, in which case $\Omega(a) = \Omega'(a) = \mathbf{I}$, or for some least $n \in \omega$, $a{\restriction}n+1 \notin \sigma$. But the construction of σ' only adds moves for **I** outside T from positions *already* outside T. That is, n is guaranteed odd, so by the construction of Ω', $\Omega'(a) = \mathbf{I}$. q.e.d

The following definition gives a concise and convenient notation for games without rules.

Definition 0.11. The notation $\mathrm{G}_X(A)$ for $A \subseteq {}^{\omega}X$ represents the game without rule $\langle {}^{<\omega}X, \Omega \rangle$ where

$$\Omega(a) \stackrel{\text{def}}{=} \begin{cases} \mathbf{I} & \text{if } a \in A, \\ \mathbf{II} & \text{otherwise.} \end{cases}$$

The notation $\mathrm{G}_X(A, B)$ for $A, B \subseteq {}^{\omega}X$ and $A \cap B = \varnothing$ represents the game without rule $\langle {}^{<\omega}X, \Omega' \rangle$ where

$$\Omega'(a) \stackrel{\text{def}}{=} \begin{cases} \mathbf{I} & \text{if } a \in A, \\ \mathbf{II} & \text{if } a \in B, \\ \mathbf{D} & \text{otherwise.} \end{cases}$$

Wherever possible we omit the subscript.

Chapter 1

Topology and labellings

This chapter lays the foundations for those to come. The foundational evaluation of complexity in this thesis is the approach taken in descriptive set theory, based on a topological analysis of the payoff set of a game. This measure historically motivated the analysis via labellings which underpins most of the results presented later.

1.1 Definitions

The heart of this analysis is a hierarchy of topological complexity known as the Borel hierarchy. The construction given can be applied on any topological space, but it has convenient properties (such as producing a hierarchy in the usual sense!) on Polish spaces.

Definition 1.1 (Topological and Polish spaces). A TOPOLOGICAL SPACE is a pair $\mathcal{S} = \langle X, \mathcal{F} \rangle$, where $\mathcal{F} \subseteq \wp(X)$ satisfies the following conditions:

1. $\varnothing, X \in \mathcal{F}$;
2. If $A, B \in \mathcal{F}$, then $A \cap B \in \mathcal{F}$;
3. For every $\mathcal{E} \subseteq \mathcal{F}$, $\bigcup \mathcal{E} \in \mathcal{F}$.

The set X is called the FIELD of $\mathcal{S}$, and $\mathcal{F}$ is a TOPOLOGY on X. For $A \in \mathcal{F}$ we call A OPEN, its complement $X \setminus A$ CLOSED. If A is both open and closed, it is CLOPEN.

A topological space is called a POLISH SPACE if it is separable and admits a compatible complete metric. We will be concerned only with the Polish spaces $\langle {}^{\omega}X, \mathcal{F} \rangle$ where X is any finite set and $\mathcal{F}$ is given by closing under conditions 1., 2. and 3. above the set $\mathcal{B}_X$ of BASIC OPEN SETS $\mathcal{B}_X \stackrel{\text{def}}{=} \{[s] \; ; \; s \in {}^{<\omega}X\}$. (It follows from this definition that a set $A \subseteq {}^{\omega}X$ is closed iff for some tree T, $A = [T]$, a characterisation that we will frequently make use of.)

Since we use always the same topology, from here on we omit it from the definitions and call ${}^{\omega}X$ the Polish space. Two of these Polish spaces deserve

special mention: the BAIRE SPACE ${}^{\omega}\omega$ is generally used in descriptive set theory for game outcome sets, and the CANTOR SPACE ${}^{\omega}2$ will often come in handy for its simplicity in cases where we want X finite. We call elements of these two spaces REALS.

For more details, see [Kec95, Chapter 1, "Polish spaces"].

Definition 1.2 (The Borel hierarchy). The BOREL HIERARCHY on a topological space $\langle X, \mathcal{S}\rangle$ starts by defining the class $\underset{\sim}{\mathbf{\Sigma}}^0_1$: for $A \subseteq X$,

$$A \in \underset{\sim}{\mathbf{\Sigma}}^0_1 \overset{\text{def}}{\iff} A \in \mathcal{S}, \text{ i.e., } A \text{ is open,}$$

and the rest of the hierarchy is defined by the following recursion:

$$\begin{aligned} A \in \underset{\sim}{\mathbf{\Pi}}^0_\alpha &\overset{\text{def}}{\iff} X \setminus A \in \underset{\sim}{\mathbf{\Sigma}}^0_\alpha \\ \underset{\sim}{\mathbf{\Delta}}^0_\alpha &\overset{\text{def}}{=} \underset{\sim}{\mathbf{\Sigma}}^0_\alpha \cap \underset{\sim}{\mathbf{\Pi}}^0_\alpha \end{aligned}$$

and for $\alpha > 1$,

$$\begin{aligned} A \in \underset{\sim}{\mathbf{\Sigma}}^0_\alpha \overset{\text{def}}{\iff}\ & \text{there is a sequence } \langle A_i \,;\, i \in \omega\rangle \\ & \text{such that each } A_i \in \bigcup_{\beta<\alpha} \underset{\sim}{\mathbf{\Pi}}^0_\beta \text{ and } A = \bigcup_{i\in\omega} A_i. \end{aligned}$$

The result is the hierarchy shown in Figure 1.1. For more details, see [Kec95, Chapter 2, "Borel Sets"].

Theorem 1.3 ([And01, Exercise 3.9]). *The Borel hierarchy when defined on a Polish space is proper until* $\underset{\sim}{\mathbf{\Sigma}}^0_{\omega_1} = \underset{\sim}{\mathbf{\Pi}}^0_{\omega_1} = \underset{\sim}{\mathbf{\Delta}}^0_{\omega_1}$.

We call the union of the countable levels of the hierarchy, , $\bigcup_{\alpha<\omega_1} \underset{\sim}{\mathbf{\Delta}}^0_1$, the "Borel sets", and denote the class of all such sets (for a given topological space X) by Borel_X. Since we are mostly concerned with the Baire space, we write "Borel" for $\mathrm{Borel}_{{}^{\omega}\omega}$.

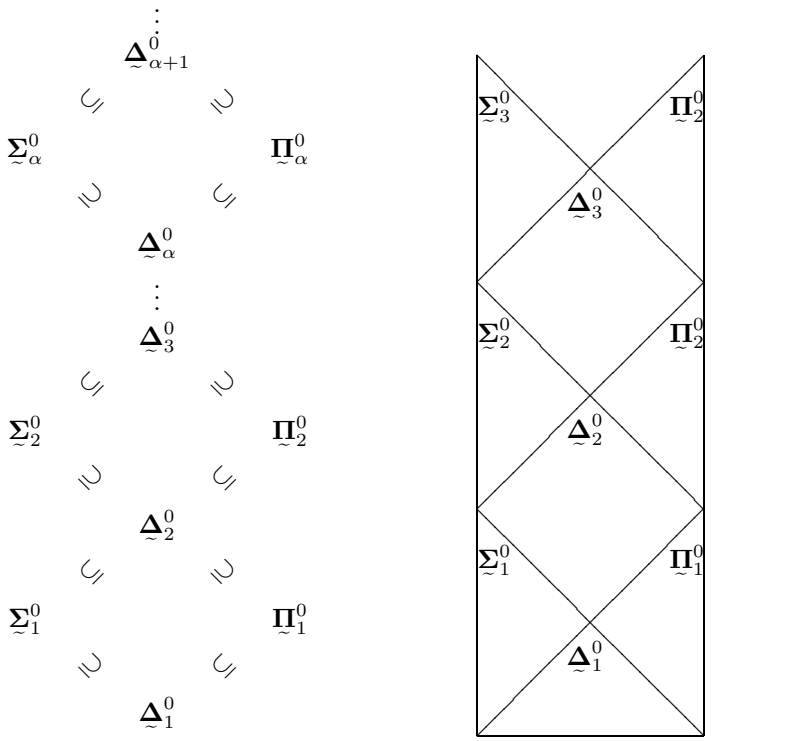
An alternative, purely syntactic, characterisation of this hierarchy will also be useful. The account given here is simplified as much as possible for our present purposes; for more details, see [Kan03, pg. 152, "The Definability Context"].

Definition 1.4 (Second-order arithmetic). SECOND-ORDER ARITHMETIC is the two-sorted structure

$$\mathcal{A}^2 \overset{\text{def}}{=} \langle \omega,\ {}^{\omega}\omega, \mathrm{ap}, +, \times, \leq, 0, 1\rangle$$

where $\mathrm{ap}\colon {}^{\omega}\omega \times \omega \to \omega$ (function application) is defined by $\mathrm{ap}(a, n) \overset{\text{def}}{=} a(n)$, and the other elements have their usual arithmetical definitions. The language has symbols for these elements, function variables a_i ranging over ${}^{\omega}\omega$ and number variables x_i ranging over ω, and corresponding function and number quantifiers $\exists^1$, $\forall^1$ for elements of ${}^{\omega}\omega$ and $\exists$, $\forall$ for elements of ω.

Terms for numbers are defined in the obvious manner, and if ϑ is a number term and Φ a formula, the BOUNDED QUANTIFIERS $\exists x_i{<}\vartheta\Phi$ and $\forall x_i{<}\vartheta\Phi$ represent respectively the formulae $\exists x_i(x_i < \vartheta \wedge \Phi)$ and $\forall x_i(x_i < \vartheta \to \Phi)$.



(a) Explicit class inclusion (b) Classes as (downward-inclusive) regions

Figure 1.1: Two visualisations of the Borel hierarchy

A hierarchy of formulae for second-order arithmetic analogous to the finite levels of the Borel hierarchy is defined as follows:

Definition 1.5 (Alternating quantifier hierarchy). Let Φ be a formula of second-order arithmetic containing only number quantifiers. Then

$$\begin{aligned}\Phi \in \Pi_0 \iff \Phi \in \Sigma_0 &\overset{\text{def}}{\iff} \text{ the only quantifiers appearing in } \Phi \text{ are bounded,}\\ \Phi \in \Sigma_{n+1} &\overset{\text{def}}{\iff} \Phi \text{ is of the form } \exists x_i \Psi \text{ where } \Psi \in \Pi_n,\\ \Phi \in \Pi_{n+1} &\overset{\text{def}}{\iff} \Phi \text{ is of the form } \forall x_i \Psi \text{ where } \Psi \in \Sigma_n.\end{aligned}$$

By various quantifier rearrangement and coding equivalences (a prenex normal form theorem for second-order logic), every formula of second-order arithmetic using only number quantifiers is equivalent to a formula in this hierarchy. Finally, the hierarchy is related to the Borel hierarchy of sets of reals by the following theorem:

Theorem 1.6 ([Kan03, Proposition 12.6]). *Take $A \subseteq {}^\omega\omega$ arbitrary. Then $A \in \mathbf{\Sigma}^0_n$ for $n \in \omega$ iff for some formula $\Phi \in \Sigma_n$ with two free function variables*

and some $b \in {}^{\omega}\omega$*, we have* $a \in A \iff \Phi(a, b)$*. Likewise for* $A \in \underset{\sim}{\mathbf{\Pi}}^0_n$*, with* $\Phi \in \Pi_n$*.*

This theorem means that we can place upper bounds on a set's location in the Borel hierarchy simply by giving a defining formula for it.

1.2 The strength of determinacy axioms

Recall that for $A \subseteq {}^{\omega}X$, $\mathrm{G}_X(A)$ is the game where players play ω moves alternately from X, and **I** wins if the result lies in A. Recall also that a game is determined if a winning strategy exists for one of the players.

For $\Gamma \subseteq \wp({}^{\omega}X)$, the DETERMINACY AXIOM $\mathsf{Det}_X(\Gamma)$ is the statement "For all $A \in \Gamma$, $\mathrm{G}_X(A)$ is determined." (As usual, we omit the subscript wherever unambiguous.) Determinacy axioms can be used to show that the Borel hierarchy is in fact measuring complexity in some natural manner.

Theorem 1.7 ([Mar75]).

$$\mathsf{ZFC} \vdash \mathsf{Det}_{\omega}(\text{Borel})$$

According to this theorem, the entire Borel hierarchy is determined. However [Fri81] showed that proving this requires uncountably many iterations of the power set operation.[1] Less powerful axiom systems than ZFC prove determinacy less far up the Borel hierarchy, showing that topological complexity corresponds in some natural fashion to proof-theoretic strength.[2]

The idea is to start with a weak base theory T and measure the proof-theoretic strength of the systems $\mathsf{T}+\mathsf{Det}(\Gamma)$, for various Γ in the Borel hierarchy. More details can be found in [HMar], here only the flavour of the results is given.

The following results can be found in Chapter 1 of [Mar]:

Theorem 1.8 (Martin). *Let* ZFC^- *be the axiom system* ZFC *without the powerset axiom.*

$$\mathsf{ZFC}^- \vdash \mathsf{Det}(\underset{\sim}{\mathbf{\Delta}}^0_4) \qquad \textit{(Corollary 1.4.12)}$$
$$\mathsf{ZFC}^- \nvdash \mathsf{Det}(\underset{\sim}{\mathbf{\Sigma}}^0_4) \qquad \textit{(Exercise 1.4.1)}$$

[HMar] discusses further results based on weakenings of SOA, an axiomatisation of the system of second-order arithmetic. By the second result of Theorem 1.8, $\mathsf{SOA} \nvdash \mathsf{Det}(\underset{\sim}{\mathbf{\Sigma}}^0_4)$. However SOA proves $\mathsf{Det}(\underset{\sim}{\mathbf{\Sigma}}^0_3)$ [HMar, via a theorem of

[1] This statement is difficult to make precise without a lot of extra definitions; indeed, an earlier draft of this thesis attempted to do so and merely misrepresented the result. Broadly speaking, Friedman starts with the axiomatisation SOA of second-order arithmetic, and shows that $\mathsf{SOA}+\mathsf{Det}_{\omega}(\text{Borel})$ proves the existence of (codes for) every countable ordinal. However SOA alone is not this powerful, so the extra proof-theoretic strength is the result of the determinacy axiom.

[2] It is perhaps worth noting that Borel determinacy also requires some fragment of the axiom of choice; ZF does not suffice, but adding the axiom of choice for countable families of sets does [Hur93].

Welch]. Even weaker systems can be constructed by limiting the comprehension scheme and induction axiom (by restricting the formulae appearing within them to low levels of the alternating quantifier hierarchy). Some of these correspond exactly to lower-level determinacy axioms, however precise statements of these results would take us too far from the main ideas of the thesis.

The crucial point is that proofs of determinacy become more complicated for higher Borel levels, and (as the results above by Friedman and Martin show) this is by necessity. The games we are concerned with, however, lie in the lowest Borel levels. We can not only prove directly that they are determined, we can even construct winning strategies witnessing this.

1.3 Labellings

The classic determinacy proof is Gale and Stewart's proof of open determinacy, $\mathsf{Det}(\underset{\sim}{\mathbf{\Sigma}}^0_1)$. The technique of labelling used for this proof is the inspiration for all the solution procedures presented here.

Definition 1.9 (Labellings). The classes

$$\begin{aligned} \mathrm{L}_{\mathbf{I}} &\stackrel{\text{def}}{=} \{\mathbf{I}_\alpha \,;\, \alpha \in \mathrm{Ord}\}, \\ \mathrm{L}_{\mathbf{D}} &\stackrel{\text{def}}{=} \{\mathbf{D}_\alpha \,;\, \alpha \in \mathrm{Ord}\}, \\ \mathrm{L}_{\mathbf{II}} &\stackrel{\text{def}}{=} \{\mathbf{II}_\alpha \,;\, \alpha \in \mathrm{Ord}\} \end{aligned}$$

are called respectively the **I**-LABELS, **D**-LABELS and **II**-LABELS.

A LABELLING on a graph $\langle V, E\rangle$ is a function

$$\ell \colon V \to \mathrm{L}_{\mathbf{I}} \cup \mathrm{L}_{\mathbf{D}} \cup \mathrm{L}_{\mathbf{II}}.$$

The set $\mathrm{ran}(\ell) \cap \mathrm{L}_{\mathbf{I}}$ is known as the **I**-LABELS OF ℓ, and likewise for **D** (the DRAW LABELS) and **II**.

If $\ell(v) = x_\alpha$ (for $x \in \{\mathbf{I}, \mathbf{D}, \mathbf{II}\}$), then v is a vertex of HEIGHT α, and we say $\mathrm{ht}_\ell(v) = \alpha$.

We define two well-orders on labels, $<_{\mathbf{I}}$ and $<_{\mathbf{II}}$, given by

$$\mathbf{I}_\alpha <_{\mathbf{I},\mathbf{II}} \mathbf{I}_\beta \stackrel{\text{def}}{\iff} \mathbf{D}_\alpha <_{\mathbf{I},\mathbf{II}} \mathbf{D}_\beta \stackrel{\text{def}}{\iff} \mathbf{II}_\alpha <_{\mathbf{I},\mathbf{II}} \mathbf{II}_\beta \stackrel{\text{def}}{\iff} \alpha < \beta$$

and for all $\alpha, \beta, \gamma \in \mathrm{Ord}$,

$$\begin{aligned} \mathbf{I}_\alpha <_{\mathbf{I}} \mathbf{D}_\beta <_{\mathbf{I}} \mathbf{II}_\gamma, \\ \mathbf{II}_\alpha <_{\mathbf{II}} \mathbf{D}_\beta <_{\mathbf{II}} \mathbf{I}_\gamma. \end{aligned}$$

The well-orders $<_{\mathbf{I}}$ and $<_{\mathbf{II}}$ are in a sense preferences: each player prefers to win if possible, draw if necessary, and lose only if unavoidable. If the intuitive meaning of the labels accurately reflects the state of affairs, a player should be able to 'read off' a strategy from a labelling by selecting moves with least (in the appropriate order) label, and the resulting strategy will be 'optimal' for that player. The following definition formalises these ideas.

Definition 1.10 (Good strategies and sound labellings). Let $G = \langle T, \Omega \rangle$ be a game, and $\ell \colon T \to \mathrm{L}_{\mathbf{I}} \cup \mathrm{L}_{\mathbf{D}} \cup \mathrm{L}_{\mathbf{II}}$ a labelling on T.

A strategy σ for player i in G is ℓ-GOOD if for all positions $s \in \sigma$ owned by i and labelled by ℓ,

$$\ell(\sigma(s)) = \min_{<_i}\{\ell(s^\frown x) \,;\, x \in X\} \text{ (or } \ell(\sigma(s)) \text{ is undefined), and}$$
$$\sigma(s) = \min\{s^\frown x \,;\, x \in X \text{ and } \ell(\sigma(s)) = \ell(s^\frown x)\}.$$

(We implicitly assume a suitable well-order on the move set X. Recall that the move set is finite, thus such an order always exists; for games on the Baire and Cantor spaces we use the usual ordering on ω.)

The labelling ℓ is G-SOUND if every ℓ-good strategy σ for player i satisfies the following two conditions:

- σ is winning for player i from positions labelled from L_i;
- σ is non-losing but not winning for player i from positions labelled $\mathbf{D}$ and owned by i.

Remark. These definitions are adapted from [LS05] but include a significant alteration: a labelling is not required to be total. Thus trivial (empty) G-sound labellings exist for any game G. The advantage of this notation is that we can speak precisely about the process of *producing* a G-sound total labelling ℓ, for instance by constructing a sequence of partial labellings $\langle \ell_\alpha \,;\, \alpha < \beta \rangle$ such that each ℓ_α is G-sound and $\ell = \bigcup\{\ell_\alpha \,;\, \alpha < \beta\}$ is both total and G-sound. The following section provides an example of such a construction.

1.4 The Gale-Stewart labelling

The Gale-Stewart procedure was introduced in [GS53] to prove $\mathsf{Det}(\mathbf{\Sigma}^0_1)$, that open sets produce determined games. This procedure is the jump-off point for most of the algorithms considered in this thesis.

The idea will be familiar to game theorists, under the name BACKWARD INDUCTION. This idea in game theory has however become somewhat problematic, since it depends on notions of rationality and rational play that turn out on closer inspection to be non-trivial.[3] In the set-theoretic context these problems are avoided by strengthening the requirements on strategies: the properties we are interested in (being "winning" or "non-losing") are required to hold against *any* play by the other player.[4]

Informally, we start with the positions generating the basic open sets (at which we know the outcome), and then 'back up' the values through the tree by observing where **I** can move to a **I**-labelled node, and where **II** is forced to do so.

[3] See for instance [dB04].

[4] The convenient existence of such powerful strategies only holds for two-player games. As shown in [LÖ3], extremely simple games with three players can be constructed that admit no winning strategies.

Definition 1.11. Let ℓ be a labelling on $T \subseteq {}^{<\omega}X$. Let $\alpha_{\mathbf{I}} \stackrel{\text{def}}{=} \sup\{\beta+1 \,;\, \mathbf{I}_\beta \in \mathrm{ran}(\ell)\}$, and similarly $\alpha_{\mathbf{II}}$ and $\alpha_{\mathbf{D}}$. Define the BACKUP of ℓ, $\mathrm{backup}(\ell) \supseteq \ell$ as follows:

$$\mathrm{backup}(\ell)(s) = \begin{cases} \ell(s) & \text{if } s \in \mathrm{dom}(\ell), \\ \mathbf{I}_{\alpha_{\mathbf{I}}} & \text{if } \mathrm{lh}(s) \text{ even and } \ell(t) \in \mathrm{L}_{\mathbf{I}} \text{ for some } t \in \mathrm{succ}_T(s), \\ \mathbf{I}_{\alpha_{\mathbf{I}}} & \text{if } \mathrm{lh}(s) \text{ odd and } \ell(t) \in \mathrm{L}_{\mathbf{I}} \text{ for all } t \in \mathrm{succ}_T(s), \\ \mathbf{II}_{\alpha_{\mathbf{II}}} & \text{if } \mathrm{lh}(s) \text{ odd and } \ell(t) \in \mathrm{L}_{\mathbf{II}} \text{ for some } t \in \mathrm{succ}_T(s), \\ \mathbf{II}_{\alpha_{\mathbf{II}}} & \text{if } \mathrm{lh}(s) \text{ even and } \ell(t) \in \mathrm{L}_{\mathbf{II}} \text{ for all } t \in \mathrm{succ}_T(s), \\ \mathbf{D}_{\alpha_{\mathbf{D}}} & \text{if } \mathrm{succ}_T(s) \subseteq \mathrm{ran}(\ell), \{t \in \mathrm{succ}_T(s) \,;\, \ell(t) \in \mathrm{L}_{\mathbf{D}}\} \neq \varnothing, \\ & \text{and } \{t \in \mathrm{succ}_T(s) \,;\, \ell(t) \in \mathrm{L}_{\mathbf{I}}\} = \varnothing \text{ if } \mathrm{lh}(s) \text{ even}, \\ & \qquad \{t \in \mathrm{succ}_T(s) \,;\, \ell(t) \in \mathrm{L}_{\mathbf{II}}\} = \varnothing \text{ if } \mathrm{lh}(s) \text{ odd}. \end{cases}$$

Theorem 1.12. *Let $G = \langle T, \Omega\rangle$ be a game, ℓ a labelling on T. If ℓ is G-sound, then so is* $\mathrm{backup}(\ell)$.

Proof. A $\mathrm{backup}(\ell)$-good strategy is necessarily ℓ-good, so we need only check the newly labelled positions.

Let σ be a $\mathrm{backup}(\ell)$-good strategy, and $s \in \sigma$ a position such that $s \in \mathrm{dom}(\mathrm{backup}(\ell)) \setminus \mathrm{dom}(\ell)$.

Suppose that $\mathrm{backup}(\ell)(s) \in \mathrm{L}_{\mathbf{I}}$ (the case for $\mathbf{II}$ is symmetric). If $\mathrm{lh}(s)$ is even, then by construction $\ell(\sigma(s)) \in \mathrm{L}_{\mathbf{I}}$, so by hypothesis σ is winning from $\sigma(s)$ (since σ is ℓ-good), therefore σ is winning from s. If, on the other hand, $\mathrm{lh}(s)$ is odd, then by construction every $s^\frown x \in T$ must be labelled $\mathbf{I}$ by ℓ, so again by hypothesis σ is winning from every successor of s so σ is winning from s.

Suppose instead that $\mathrm{backup}(\ell)(s) \in \mathrm{L}_{\mathbf{D}}$. By construction, if s is owned by player i then no successor of s is labelled from L_i by ℓ, and all successors of s are labelled. Since σ is ℓ-good, no winning strategy for i from s exists. Furthermore, since σ is $\mathrm{backup}(\ell)$-good, a successor labelled $\mathbf{D}$ by ℓ will be chosen. Finally, since ℓ is G-sound, this successor is not winning for the opponent, so σ is non-losing for i from s.

Thus $\mathrm{backup}(\ell)$ is G-sound if ℓ is. q.e.d

Definition 1.13. The GALE-STEWART LABELLING PROCEDURE labels a tree $T \subseteq {}^{<\omega}X$ from a SEED $B \subseteq {}^{<\omega}X$ as follows:

$$\ell_0(s) = \begin{cases} \mathbf{I}_0 & \text{if } \exists n{\in}\omega(s{\restriction}n \in B), \\ \bot & \text{otherwise.} \end{cases}$$

$$\ell_{\alpha+1} = \mathrm{backup}(\ell_\alpha),$$

$$\ell_\lambda = \bigcup\{\ell_\alpha \,;\, \alpha < \lambda\} \text{ for } \lambda \text{ limit.}$$

Let γ be the least fixpoint of this procedure, then the total labelling ℓ_{GS} is defined by

$$\ell_{\mathrm{GS}}(s) = \begin{cases} \ell_\gamma(s) & \text{if this is defined,} \\ \mathbf{II}_0 & \text{otherwise.} \end{cases}$$

Lemma 1.14. *The fixpoint γ exists.*

[Any stage that does not label a previously unlabelled position is the fixpoint. There are only countably many positions to label.]

Theorem 1.15. *Let $A \subseteq {}^{\omega}X$ be an open set with basis B. The Gale-Stewart labelling ℓ_{GS} with seed B is $\mathrm{G}(A)$-sound.*

Proof. First, the labelling ℓ_0 is $\mathrm{G}(A)$-sound. Let $a \in {}^{<\omega}X$ be arbitrary such that for some $n \in \omega$, $\ell_0(a\restriction n) = \mathbf{I}_0$. Then for some $s \in B$, $s \subseteq a$, so since B is a basis for A, $a \in A$. That is, for any position s labelled $\mathbf{I}_0$ by ℓ_0, $\mathbf{I}$ has *already won* any play passing through s, so all strategies (and thus all ℓ_0-good strategies) are winning for $\mathbf{I}$ at s.

Next, Theorem 1.12 guarantees that all successor-level labellings are $\mathrm{G}(A)$-sound, so long as limit-level labellings are. So let $\lambda \in \mathrm{Ord}$ be limit such that for all $\alpha < \lambda$, ℓ_α is $\mathrm{G}(A)$-sound. Any position s labelled such that $\ell_\lambda(s) \in \mathrm{L}_{\mathbf{I}}$ received the label at some previous stage, say α. But by the inductive hypothesis, ℓ_α was $\mathrm{G}(A)$-sound, that is, any ℓ_α-good strategy is winning for $\mathbf{I}$ at s. And if a strategy σ is ℓ_λ-good, then it is also ℓ_α-good for all $\alpha < \lambda$.

That is, both successor and limit stage labellings are $\mathrm{G}(A)$-sound, so we need only check soundness of the **II**-labels of ℓ_{GS}.

Let T be the tree generated from A (i.e., $T \stackrel{\mathrm{def}}{=} \{a\restriction n \ ; \ a \in A, n \in \omega\}$). Let σ be an ℓ_{GS}-good strategy for $\mathbf{II}$ in $\mathrm{G}(A)$ (so that $\sigma \subseteq T$), and let $s \in \sigma$ be arbitrary such that $\ell_{\mathrm{GS}}(s) = \mathbf{II}_0$.

Assume toward a contradiction that for some $a \in [\sigma]$ such that $s \subseteq a$, $a \in A$ (that is, σ is not winning from s for $\mathbf{II}$).

Then since B is a basis for A, for some $t \in B$ we have $t \subseteq a$. It cannot be that $t \subseteq s$, since then $\ell_0(s) = \mathbf{I}_0$, so we have $s \subsetneq t$.

Now we show, towards a contradiction, that at some stage $\alpha < \gamma$, $\ell_\alpha(s) = \mathbf{I}_\alpha$. The proof is by induction along $t \setminus s$, showing that each $t(n)$ for $n \geq \mathrm{lh}(s) - 1$ is labelled $\mathbf{I}$ at some stage before γ. The position t provides a base case.

For the inductive step, let $u \in \sigma$ be arbitrary such that for some $\alpha < \gamma$, $\ell_\alpha(u) \in \mathrm{L}_{\mathbf{I}}$. Let $v = u\restriction(\mathrm{lh}(u) - 1)$ be the predecessor of u.

Suppose v is owned by $\mathbf{I}$. Then $\ell_{\alpha+1}(v) \in \mathrm{L}_{\mathbf{I}}$. Suppose instead that v is owned by $\mathbf{II}$. Then since σ is ℓ_{GS}-good, for all successors $w \in \mathrm{succ}_T(v)$, $\ell_\gamma(w) \in \mathrm{L}_{\mathbf{I}}$ (otherwise for some $w \in \mathrm{succ}_T(v)$, $\ell_{\mathrm{GS}}(w) \in \mathrm{L}_{\mathbf{II}}$ so σ chooses w instead of u). Let $\beta = \sup\{\alpha \ ; \ w \in \mathrm{succ}_T(v), \ell_\gamma(w) = \mathbf{I}_\alpha)\}$. Then if β is limit, $\ell_{\beta+1}(v) = \mathbf{I}_{\beta+1}$, otherwise $\ell_\beta(v) = \mathbf{I}_\beta$. This completes the inductive step.

By this induction, for no $t \in \sigma$ such that $t \supseteq s$ can it be the case that $\ell_{\mathrm{GS}}(t) \in \mathrm{L}_{\mathbf{I}}$ while $\ell_{\mathrm{GS}}(s) \in \mathrm{L}_{\mathbf{II}}$. So σ is winning for $\mathbf{II}$ from all **II**-labelled positions.

We already showed that σ is winning for $\mathbf{I}$ from all positions labelled at successor and limit stages. So we have that σ is winning for player i from positions labelled i. Since σ was arbitrary ℓ_{GS}-good, this proves that ℓ_{GS} is $\mathrm{G}(A)$-sound. q.e.d

Remark. The original proof that a strategy produced by the Gale-Stewart labelling procedure is winning was substantially different. It used the observation

that such a strategy (for **I**) from a position labelled **I** moves always to a **I**-position labelled in a preceding stage. Since the ordinals are well-founded, a play according to such a strategy eventually passes through a position labelled $\mathbf{I}_0$, confirming a win. For **II** it suffices to avoid forever positions labelled **I**, in much the same way as the proof given above.

The more general construction via backup and soundness of labellings is given here because these building blocks will be reused in later sections.

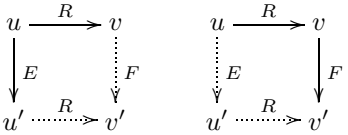
1.5 Combinatorial labellings

Combinatorial labellings are introduced in [LS05]. The idea is to specify those labellings that are both sound (for their respective games) and are derived 'bottom-up' from the structure of the tree.

Definition 1.16 (Bisimulation). Let $\langle U, E\rangle$ and $\langle V, F\rangle$ be directed graphs. A relation $R \subseteq U \times V$ is a BISIMULATION if the following two conditions hold:

$\forall u, u' \in U \forall v \in V$(if uEu' and uRv, then for some $v' \in V$, vFv' and $u'Rv'$),
$\forall u \in U \forall v, v' \in V$(if uRv and vFv', then for some $u' \in U$, uEu' and $u'Rv'$).

This is made clearer by a diagram, in which the dotted arrows are required by the definition:



If we treat the graphs as transition systems then R is a relation under which either system can simulate the other.

Definition 1.17. Let $T \subseteq {}^{<\omega}X$ a tree and $A \subseteq {}^{\omega}X$ be arbitrary, and $R \subseteq T \times T$ a bisimulation on the positions in T. We say R PRESERVES A if for all $a, b \in [T]$ we have

$$\text{if } \forall n \in \omega(\langle a{\restriction}n, b{\restriction}n\rangle \in R) \text{ then } a \in A \iff b \in A.$$

We extend this to a game $\langle T, \Omega\rangle$ by requiring that R preserve $\Omega_{\mathbf{I}}$, $\Omega_{\mathbf{D}}$ and $\Omega_{\mathbf{II}}$, and say that R preserves Ω.

If $s, t \in T$ are positions both owned by the same player, we say they are Ω-bisimilar if there exists an Ω-preserving bisimulation R such that sRt.

A combinatorial ('bottom-up') labelling should respect bisimulation. Intuitively, if two positions are bisimilar then the trees below them 'look the same' from the point of view of playing the game. A bottom-up labelling should thus not distinguish between them.

Definition 1.18. A labelling ℓ for the tree T is Ω-COMBINATORIAL if for every $s, t \in T$ that are Ω-bisimilar, $\ell(s) = \ell(t)$.

Theorem 1.19 ([LS05, Proposition 4.2]). *There is a $\mathbf{\Sigma}^0_2$ set $A \subseteq {}^\omega 2$ such that no $\mathrm{G}(A)$-sound labelling is A-combinatorial.*

Proof. Define A by

$$a \in A \overset{\text{def}}{\iff} \exists n{\in}\omega \forall m{\geq}n(a(m) = 0).$$

Clearly A is $\mathbf{\Sigma}^0_2$, and **II** has many winning strategies ("move always 1" is a very simple one). But by observation, any two $s, s' \in {}^\omega 2$ owned by the same player are A-bisimilar. (Take the bisimulation $\{\langle s^\frown t, s'^\frown t\rangle \,;\, t \in {}^\omega 2\}$ as witness.) This means that any A-combinatorial labelling ℓ must give every position with odd length the same label. And in that case, a strategy for **II** that is ℓ-good will make **II** always move 0. q.e.d

In fact, [LS05] shows that $\mathbf{\Sigma}^0_2$ is the first Borel class containing a game that does not admit a combinatorial labelling:

Theorem 1.20 ([LS05, Theorem 6.5]). *Every $\mathbf{\Delta}^0_2$ set A has a total $\mathrm{G}(A)$-sound and A-combinatorial labelling.*

Proof sketch. The proof proceeds by constructing $\mathrm{G}(A)$-sound and A-combinatorial labellings for sets A in the Hausdorff difference hierarchy. By a theorem of Hausdorff and Kuratowski, $\mathbf{\Delta}^0_1$ is the union of all countable-level Hausdorff difference classes. q.e.d

1.6 The combinatorial game

Definition 1.21. The COMBINATORIAL GAME on a tree $T \subseteq {}^{<\omega}X$ is the rulification of the game $\langle T, \Omega\rangle$ where $\Omega_{\mathbf{D}} = [T]$.

That is, plays within the tree are a draw, otherwise the first player to exit the tree loses. (Note that this is therefore a game $\mathrm{G}(A, B)$ with $A, B \subseteq {}^\omega X$ both open.)

The obvious extension of the Gale-Stewart procedure suffices to label a tree for solving the combinatorial game.

Definition 1.22. The COMBINATORIAL GALE-STEWART PROCEDURE labels a tree $T \subseteq {}^{<\omega}X$ according to two seeds $B, C \subseteq {}^{<\omega}X$ as follows:

$$\ell_0(s) = \begin{cases} \mathbf{I}_0 & \text{if } \exists n{\in}\omega(s{\restriction}n \in B), \\ \mathbf{II}_0 & \text{if } \exists n{\in}\omega(s{\restriction}n \in C), \\ \bot & \text{otherwise.} \end{cases}$$

$$\ell_{\alpha+1} = \mathrm{backup}(\ell_\alpha),$$

$$\ell_\lambda = \bigcup\{\ell_\alpha \,;\, \alpha < \lambda\} \text{ for } \lambda \text{ limit.}$$

Let γ be the fixpoint of this procedure, then the total labelling ℓ_{CGS} is defined by

$$\ell_{\mathrm{CGS}}(s) = \begin{cases} \ell_\gamma(s) & \text{if this is defined,} \\ \mathbf{D}_0 & \text{otherwise.} \end{cases}$$

Remark. This definition can only be applied if B and C are 'strongly disjoint' in the following sense: $\forall s{\in}B \forall t{\in}C(\neg(s \subseteq t \vee t \subseteq s))$. This condition is obviously satisfied by the seeds produced by the combinatorial game, which is all we need for the present purposes.

Theorem 1.23. *Let* $\mathrm{G}(A, B)$ *be a combinatorial game, the rulification of a tree* $T \subseteq {}^{<\omega}X$*, and let* $C, D \subseteq {}^{<\omega}X$ *be bases for* A *and* B*, respectively. Then the combinatorial Gale-Stewart labelling* ℓ_{CGS} *with seeds* C *and* D *is* $\mathrm{G}(A, B)$*-sound.*

Proof. The proof for positions labelled **I** and **II** is analagous to that for Theorem 1.15.

Let $s \in {}^{<\omega}X$ be a position such that $\ell_{\mathrm{CGS}}(s) = \mathbf{D}_0$, and let γ be the fixpoint of the sequence of backup labellings. Without loss of generality, let $\mathrm{lh}(s)$ be even (the proof for positions owned by **II** is analagous). It suffices to show two things: no successor of s is labelled **I**, and some successor of s is labelled **D**.

(These two requirements ensure that an ℓ_{CGS}-good strategy for **I** is drawing, while the absence of a winning strategy is shown by the same argument as demonstrated the soundness of a **II**-label given by ℓ_{GS}: if such a strategy existed, s would be labelled **I** before ℓ_γ.)

But the two requirements follow directly from the definition of backup: since s is owned by **I**, if some successor t becomes labelled **I** at stage α, then $\ell_{\alpha+1}(s) = \mathbf{I}$. Likewise, if all successors are labelled **II** at stages before γ, then $\ell_\gamma(s) = \mathbf{II}$. q.e.d

However there is another labelling procedure that also suffices. Again, the definition is given here mainly so we may make use of variations on this idea in later chapters.[5]

Definition 1.24. The TWO-PASS labelling ℓ_{twice} for a combinatorial game $\mathrm{G}(A, B)$ on ${}^{<\omega}X$ is given by the following definition: first, let $B' = \{\langle 0\rangle^\frown s \,;\, s \in B\}$. Now take $\ell_A = \ell_{\mathrm{GS}}$ applied to $\mathrm{G}(A)$, and $\ell_{B'} = \ell_{\mathrm{GS}}$ applied to $\mathrm{G}(B')$

$$\ell_{\mathrm{twice}}(s) \stackrel{\mathrm{def}}{=} \begin{cases} \ell_A(s) & \text{if } \ell_A(s) \in \mathrm{L}_{\mathbf{I}}, \\ \mathbf{II}_\alpha & \text{if } \ell_B(\langle 0\rangle^\frown s) = \mathbf{I}_\alpha, \\ \mathbf{D}_0 & \text{otherwise.} \end{cases}$$

In essence, we analyse the two open games given by A and B to get the positions winning for **I** and **II**, and any position lying in the closed portion of both these games is drawing.

Theorem 1.25. *For any combinatorial game* $\mathrm{G}(A, B)$*, the two-pass labelling* ℓ_{twice} *is* $\mathrm{G}(A, B)$*-sound.*

Proof. Let σ be an ℓ_{twice}-good strategy. Clearly, σ is also ℓ_A-good. Let $s \in \sigma$ be labelled **I**. Then since ℓ_A is $\mathrm{G}(A)$-sound, $[\sigma] \cap [s] \in A$.

[5] This labelling procedure is based on a suggestion by Be Birchall in a 'mini-paper' for the Master of Logic course "Logic and Games".

Suppose instead that $\ell_{\text{twice}}(s) \in \mathrm{L}_{\mathbf{II}}$. Then let $\sigma' = \{\langle 0\rangle^\frown t \ ; \ t \in s\}$ be a strategy for $\mathbf{I}$ in $\mathrm{G}(B')$. Clearly, σ' is $\ell_{B'}$-good, and $\ell_{B'}$ is $\mathrm{G}(B')$-sound. Since $\ell_{B'}(\langle 0\rangle^\frown s) \in \mathrm{L}_{\mathbf{I}}$, $\sigma' \cap [\langle 0\rangle^\frown s] \in B'$. But then by construction, $[\sigma] \cap [s] \in B$.

Finally, take $s \in \sigma$ such that $\ell_{\text{twice}}(s) = \mathbf{D}_0$. This means that both $\ell_A(s) = \mathbf{II}$, and $\ell_{B'}(s) = \mathbf{II}$.

Since again σ is $\mathrm{G}(A)$-good, we have $[\sigma] \cap [s] \subseteq {}^\omega X \setminus A$.

Since $\ell_{B'}(\langle 0\rangle^\frown s) = \mathbf{II}$, $[\sigma'] \cap [\langle 0\rangle^\frown s] \subseteq {}^\omega X \setminus B'$, that is, $[\sigma] \cap s \subseteq {}^\omega X \setminus B$.

Putting these two together gives us $[\sigma] \cap [s] \subseteq {}^\omega X \setminus (A \cup B)$, that is, σ is a drawing strategy.

All that remains is to show that no winning strategy from s exists when $\ell_{\text{twice}}(s) = \mathbf{D}_0$. Now suppose without loss of generality that $\mathrm{lh}(s)$ is even (the case for positions owned by $\mathbf{II}$ is analagous, but complicated by the extra first move, so we shall omit it). Note that a winning strategy for $\mathbf{I}$ in $\mathrm{G}(A, B)$ also wins $\mathrm{G}(A)$. But σ is ℓ_A-good, and $\ell_A(s) = \mathbf{II}$, so no winning strategy for $\mathbf{I}$ from s in $\mathrm{G}(A)$ exists, i.e., no winning strategy for $\mathbf{I}$ from s in $\mathrm{G}(A, B)$ exists.

Putting all this together means that ℓ_{twice} is $\mathrm{G}(A, B)$-sound. q.e.d

Chapter 2

Automata games

In this chapter I consider a number of classes of games played on finite graphs, taken from the automata theory literature. These games allow for a new complexity analysis, roughly speaking "How difficult is it to play according to a given strategy?" Specifically, how much extra information beyond the structure of the graph does the player need to remember, in order to play according to a given strategy?

2.1 Definitions

An automaton is essentially a graph with the edges labelled. Conceptually, the vertices of the graph represent configurations of a machine, and an labelled edge represents a change from one configuration (or state) to another, driven by the input of the machine.

Definition 2.1 (Automata). An AUTOMATON $\mathcal{A}$ over a finite alphabet Σ is a structure $\langle Q, q_0, \delta\rangle$ where $Q = \{q_0, q_1, \ldots, q_n\}$ is a finite set of STATES, q_0 is the START STATE, and $\delta\colon Q \times \Sigma \to Q$ is the TRANSITION FUNCTION.

The GRAPH of $\mathcal{A}$ is $\langle Q, E\rangle$ where $E \subseteq Q \times Q$ is given by

$$\langle q, q'\rangle \in E \overset{\text{def}}{\iff} \exists a{\in}\Sigma(\delta(q, a) = q').$$

We represent an automaton pictorially by drawing its graph and labelling the edges with the appropriate inputs to the transition function. Figure 2.1 shows a simple example.

Let $\mathcal{A} = \langle Q, q_0, \delta\rangle$ be an arbitrary automaton. Intuitively the automaton notion represents a machine with moving parts, which changes its configuration from state to state in response to input.

The RESPONSE of $\mathcal{A}$ to a nonempty sequence ($\text{response}_{\mathcal{A}}\colon {}^{<\omega}\Sigma \to Q$) is defined inductively as follows:

$$\begin{aligned}\text{response}_{\mathcal{A}}(\langle x\rangle) &= \delta(q_0, x),\\ \text{response}_{\mathcal{A}}(s^\frown x) &= \delta(\text{response}_{\mathcal{A}}(s), x).\end{aligned}$$

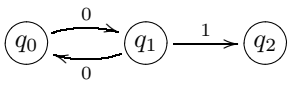


Figure 2.1: An automaton $\langle Q, q_0, \delta\rangle$ with state set $\{q_0, q_1, q_2\}$ and transition function $\big\{\langle q_0, 0, q_1\rangle, \langle q_1, 0, q_0\rangle, \langle q_1, 1, q_2\rangle\big\}$.

The TRACE of a (possibly infinite) sequence $s \in {}^{\leq\omega}\Sigma$ is given by

$$\mathrm{trace}_{\mathcal{A}}(s) = \langle \mathrm{response}_{\mathcal{A}}(s{\restriction}n)\ ;\ 0 < n \leq \mathrm{lh}(s)\rangle.$$

In both cases we omit the subscript wherever unambiguous. We also say $\mathrm{response}(s) = \mathrm{trace}(s) = \perp$ if δ is undefined on some necessary input for the induction.

Definition 2.2 (Languages). A LANGUAGE L over a finite alphabet Σ is a set of finite sequences with elements from Σ, i.e., $L \subseteq {}^{<\omega}\Sigma$. An ω-LANGUAGE is a set of *infinite* sequences $L_\omega \subseteq {}^{\omega}\Sigma$, and its elements are known as ω-WORDS.

An automaton $\mathcal{A}$ ACCEPTS a finite sequence s if $\mathrm{trace}_{\mathcal{A}}(s) \neq \perp$.

An ω-AUTOMATON is a pair $\langle \mathcal{A}, \Omega\rangle$ where $\mathcal{A}$ is an automaton with state set Q and $\Omega \subseteq {}^{\omega}Q$ is the ACCEPTANCE CONDITION. An ω-word a is accepted by an ω-automaton $\langle \mathcal{A}, \Omega\rangle$ if for all $n \in \omega$ $\mathcal{A}$ accepts $a{\restriction}n$, and if $\mathrm{trace}(a) \in \Omega$.

We call the set of finite sequences accepted by an automaton $\mathcal{A}$ the LANGUAGE GENERATED BY $\mathcal{A}$, and the ω-words accepted by $\langle \mathcal{A}, \Omega\rangle$ the ω-LANGUAGE GENERATED BY $\langle \mathcal{A}, \Omega\rangle$, denoted by $\mathcal{L}(\mathcal{A}, \Omega)$.

2.2 Standard acceptance conditions

In this section I give some standard acceptance conditions that can be defined in a natural way on the graph of the automaton, and show their complexity with respect to the Borel hierarchy. All conditions are given with respect to an automaton $\mathcal{A} = \langle Q, q_0, \delta\rangle$.

2.2.1 Reachability

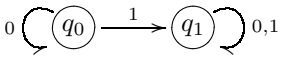
Let $Q_{\mathrm{T}} \subseteq Q$ be the TARGET SET. The acceptance condition Ω_{R} is the set of all (infinite) traces that pass though an element of Q_{T}:

Definition 2.3.

$$\Omega_{\mathrm{R}} \stackrel{\mathrm{def}}{=} \{a \in {}^{\omega}Q\ ;\ \exists n{\in}\omega(a(n) \in Q_{\mathrm{T}})\}.$$

Theorem 2.4. *For all reachability condition ω-automata $\langle \mathcal{A}, \Omega_{\mathrm{R}}\rangle$, $\mathcal{L}(\mathcal{A}, \Omega_{\mathrm{R}}) \in \mathbf{\Sigma}^0_1$. Furthermore, there exists such an automaton with $\mathcal{L}(\mathcal{A}, \Omega_{\mathrm{R}}) \notin \mathbf{\Delta}^0_1$.*

Proof. The basis of finite sequences whose last element is in Q_{T} generates Ω_{R}. For the second statement, let $\mathcal{A}$ be given by



with $Q_{\mathrm{T}} = \{q_1\}$. Let $\mathcal{L} = \mathcal{L}(\mathcal{A}, \Omega_{\mathrm{R}})$ for brevity. If $\mathcal{L}$ is closed, then for some tree $T \subseteq {}^{<\omega}2$, $\mathcal{L} = [T]$. For each $n \in \omega$, $\langle 0^n 1^\omega \rangle \in \mathcal{L}$. Since T is closed under initial subsequence, this means that for all $n \in \omega$, $\langle 0^n \rangle \in T$. But then $\langle 0^\omega \rangle \in [T]$, and $\langle 0^\omega \rangle \notin \mathcal{L}$. Thus no such T exists. q.e.d

2.2.2 The Büchi condition

Let Q_{T} be the target set. The acceptance condition Ω_{B} is the set of all infinite traces that pass *infinitely often* through an element of Q_{T}.

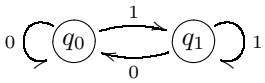
Definition 2.5.

$$\Omega_{\mathrm{B}} \stackrel{\mathrm{def}}{=} \{a \in {}^{\omega}Q \; ; \; \forall n{\in}\omega \exists m{>}n (a(m) \in Q_{\mathrm{T}})\}.$$

Theorem 2.6. *For all Büchi condition automata $\langle \mathcal{A}, \Omega_{\mathrm{B}} \rangle$, $\mathcal{L}(\mathcal{A}, \Omega_{\mathrm{B}}) \in \mathbf{\Pi}^0_2$. Furthermore, there exists such an automaton with $\mathcal{L}(\mathcal{A}, \Omega_{\mathrm{B}}) \notin \mathbf{\Sigma}^0_2$.*

Proof. That $\mathcal{L}(\mathcal{A}, \Omega_{\mathrm{B}}) \in \mathbf{\Pi}^0_2$ for any Büchi automaton can be seen from an alternating quantifiers argument. The two quantifiers in the defining formula bound the depth to which the trace has to be evaluated, and checking the membership in Q_{T} obviously only requires bounded quantifiers.

For the second statement, take the very simple automaton with alphabet $\{0, 1\}$ given by the picture



and let $Q_{\mathrm{T}} = \{q_1\}$. Let $\mathcal{L} = \mathcal{L}(\mathcal{A}, \Omega_{\mathrm{B}})$. Suppose towards a contradiction that $\mathcal{L} \in \mathbf{\Sigma}^0_2$. Then for some sequence $\langle A_n \in \mathbf{\Pi}^0_1 \; ; \; n < \omega \rangle$, $\bigcup \{A_n \; ; \; n \in \omega\} = \mathcal{L}$. Now we use a diagonalisation argument to construct an element a that does not appear in $\bigcup \{A_n \; ; \; n \in \omega\}$, but such that $a \in \mathcal{L}$.

First note that each $A_n = [T_n]$, for suitable trees $T_n \subseteq {}^{<\omega}2$. We will find an increasing family of sequences $\langle s_n \in {}^{<\omega}X \; ; \; n \in \omega \rangle$ such that for each n, $s_n \subsetneq s_{n+1}$ but also for each n, $s_n \notin T_n$. This guarantees that $a = \bigcup \{s_n \; ; \; n \in \omega\} \notin \bigcup \{A_n \; ; \; n \in \omega\}$, however at the same time we will ensure that $\mathrm{trace}(a) \in \Omega_{\mathrm{B}}$.

Let $m_0 \in \omega$ be least such that $\langle 0^{m_0} \rangle \notin T_0$. Such an m_0 exists, otherwise $\langle 0^\omega \rangle \in [T_n]$ and the trace through this sequence visits q_1 only finitely many times. Since T_n is a tree, closed under initial subsequence, $\langle 0^m 1 \rangle \notin T_n$. Let $s_0 = \langle 0^m 1 \rangle$.

Now for the inductive step: let $m_{n+1} \in \omega$ be least such that $s_n{}^\frown\langle 0^{m_{n+1}} \rangle \notin T_{n+1}$. Again, such an m_{n+1} exists, because otherwise $s_n{}^\frown\langle 0^\omega \rangle \in [T]$. Then set $s_{n+1} = s_n{}^\frown\langle 0^{m_{n+1}} 1 \rangle$.

Finally, take $a = \bigcup \{s_n \; ; \; n \in \omega\}$. Since for each n, $s_{n+1} \supsetneq s_n$, we have $a \in {}^{\omega}2$. In addition, since each s_n ends with a 1, q_1 is visited infinitely often in $\mathrm{trace}(a)$. But then $a \in \mathcal{L}$, while the construction ensures that

$$a \notin \bigcup \{[T_n] \; ; \; n \in \omega\}.$$

This contradiction shows that such a sequence of closed sets such that $\bigcup \langle A_n \rangle = \mathcal{L}(\mathcal{A}, \Omega_{\mathrm{B}})$ cannot be found. That is, for this automaton, $\mathcal{L}(\mathcal{A}, \Omega_{\mathrm{B}}) \notin \mathbf{\Sigma}^0_2$. q.e.d

2.2.3 The Muller condition

Let $Q_{\mathrm{T}} \subseteq \wp(Q)$ be a collection of target sets. The acceptance condition Ω_{M} requires that an accepted trace visit all and only the elements of some target set infinitely often.

Definition 2.7. Let $a \in {}^{\omega}X$ be an arbitrary infinite sequence. The set $\mathrm{Fin}(a) \subseteq X$ collects all elements from a occurring only finitely many times, while $\mathrm{Inf}(a) \subseteq X$ collects those elements occurring infinitely many times.

$$\mathrm{Inf}(a) \stackrel{\mathrm{def}}{=} \{x \in X \,;\, \forall n{\in}\omega \exists m{>}n (a(m) = x)\},$$
$$\mathrm{Fin}(a) \stackrel{\mathrm{def}}{=} \{x \in X \,;\, \exists n{\in}\omega \forall m{>}n (a(m) \neq x)\}.$$

Definition 2.8.

$$\Omega_{\mathrm{M}} \stackrel{\mathrm{def}}{=} \{a \in {}^{\omega}Q \,;\, \exists P \in Q_{\mathrm{T}} (\mathrm{Inf}(a) = P)\}.$$

Theorem 2.9. *For all Muller condition ω-automata $\langle \mathcal{A}, \Omega_{\mathrm{M}} \rangle$, $\mathcal{L}(\mathcal{A}, \Omega_{\mathrm{M}}) \in \mathbf{\Delta}^0_3$. Furthermore, there exists such an automaton with $\mathcal{L}(\mathcal{A}, \Omega_{\mathrm{M}}) \notin \mathbf{\Sigma}^0_2 \cup \mathbf{\Pi}^0_2$.*

Proof. Suppose first that $Q_{\mathrm{T}} = \{S\}$ is a singleton.

For $A \subseteq Q$, Let $\Phi(a, A)$ represent the formula

$$\exists n{\in}\omega \forall m{>}n (\mathrm{response}(a{\restriction}m) \in A).$$

Then $\Phi(a, S)$ is a $\mathbf{\Sigma}^0_2$ formula specifying that $\mathrm{Inf}(\mathrm{trace}(a)) \subseteq S$. As before, taking traces and testing membership in S can be done using only bounded quantifiers.[1]

Then for each $A \subsetneq S$, $\neg\Phi(a, A)$ is $\mathbf{\Pi}^0_2$. And

$$\Phi(a, S) \wedge \bigwedge_{A \subsetneq S} \neg\Phi(a, A)$$

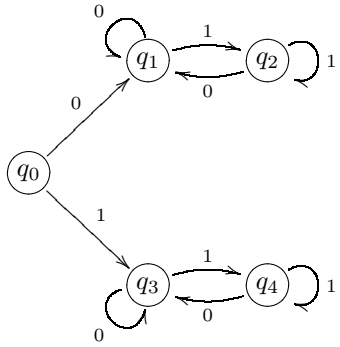
is a $\mathbf{\Delta}^0_3$ formula specifying precisely that $a \in \Omega_{\mathrm{M}}$. In case Q_{T} is not a singleton, we have only a finite disjunction of such formulae, so the result is still $\mathbf{\Delta}^0_3$.

To see that $\mathbf{\Sigma}^0_2$ does not suffice, we can take the construction given in the proof of Theorem 2.6 (the Borel height of Büchi conditions). If we take a singleton condition $\Omega_{\mathbf{\Sigma}}$ with target sets $Q_{\mathbf{\Sigma}} \stackrel{\mathrm{def}}{=} \{\{q_0, q_1\}\}$, exactly the same construction shows that any $\mathbf{\Sigma}^0_2$ set either misses an element a that we want (with $\mathrm{Inf}(\mathrm{trace}(a)) = \{q_0, q_1\}$) or includes one we don't want (with $\mathrm{Inf}(\mathrm{trace}(a)) = \{q_0\}$).

[1] In fact the language generated by an automaton condition belongs to the same Borel class as the automaton condition itself: Borel classes are boldface pointclasses, closed under taking continuous preimages, and the trace function is continuous.

To see that $\underset{\sim}{\mathbf{\Pi}}^0_2$ does not suffice, take the same automaton but with the Muller condition $\Omega_{\mathbf{\Pi}}$ given by the target sets $Q_{\mathbf{\Pi}} \stackrel{\text{def}}{=} \{\{q_1\}\}$. If this is $\underset{\sim}{\mathbf{\Pi}}^0_2$, then its negation is $\underset{\sim}{\mathbf{\Sigma}}^0_2$. But the negation of this condition states precisely that q_0 appears infinitely often in trace(a), the Büchi condition $\{q_0\}$, and we showed above that this is not $\underset{\sim}{\mathbf{\Sigma}}^0_2$.

So we have two Muller conditions, one not in $\underset{\sim}{\mathbf{\Sigma}}^0_2$ and one not in $\underset{\sim}{\mathbf{\Pi}}^0_2$. But we can combine them into a single automaton, in the following simple manner, as in the following picture.



The states of the two automata are renamed so that they are disjoint. The first transition of the new automaton selects which of the sub-automata to use. The Muller condition $\Omega_{\mathbf{\Delta}}$ is defined from the union of the two sub-automata conditions, target sets $\{\{q_1, q_2\}, \{q_3\}\}$. If this is $\underset{\sim}{\mathbf{\Sigma}}^0_2$ *or* $\underset{\sim}{\mathbf{\Pi}}^0_2$, then the relevant condition can be extracted simply by looking at the first element of the trace:

$$\text{trace}(a) \in \Omega_{\mathbf{\Sigma}} \iff \text{trace}(a) \in \Omega_{\mathbf{\Delta}} \wedge a(0) = 0,$$
$$\text{trace}(a) \in \Omega_{\mathbf{\Pi}} \iff \text{trace}(a) \in \Omega_{\mathbf{\Delta}} \wedge a(0) = 1.$$

Since the test on the first element is quantifier-free, either of the two conditions can be reduced to this one without changing Borel classes. To avoid contradiction, $\Omega_{\mathbf{\Delta}} \notin \underset{\sim}{\mathbf{\Sigma}}^0_2 \cup \underset{\sim}{\mathbf{\Delta}}^0_2$. q.e.d

2.2.4 Summary

The conditions considered have been chosen because they make a relatively neat and tidy hierarchy according to topological complexity. Figure 2.2 summarises the situation. (The positions shown are the highest extent of the relevant conditions, of course simple instances of complex conditions also exist.)

2.3 Automata games and strategies

The notion of automata games comes from picturing an automaton as a machine processing input from the environment. Player **I** takes the role of a controller, while **II** represents the environment, and the aim of the controller is to ensure that some condition on the operation of the machine holds on an infinite run (the acceptance condition of the automaton). However, the controller knows only

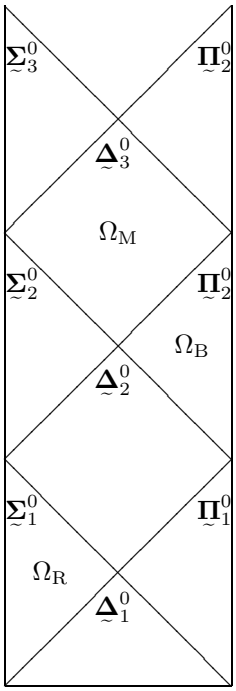


Figure 2.2: Highest positions of automata conditions in the Borel hierarchy

about the state of the machine, not about the particular moves the environment plays. These notions are formalised as follows:

Definition 2.10 (Automata games). An ALTERNATING AUTOMATON is an structure

$$\langle Q, q_0, Q_{\mathbf{I}}, Q_{\mathbf{II}}, \delta \rangle$$

where $\langle Q, q_0, \delta \rangle$ is an automaton, such that $Q_{\mathbf{I}}$ and $Q_{\mathbf{II}}$ partition Q, $q_0 \in Q_{\mathbf{I}}$, and for all $x \in \Sigma$, if $q \in Q_{\mathbf{I}}$ then $\delta(q, x) \in Q_{\mathbf{II}}$, and if $q \in Q_{\mathbf{II}}$ then $\delta(q, x) \in Q_{\mathbf{I}}$ (where these are defined).

The AUTOMATON GAME for an alternating ω-automaton $\langle \mathcal{A}, \Omega \rangle$ is the game $\langle T, A \rangle$ where $T \subseteq {}^{<\omega}\Sigma$ and $A \subseteq [T]$ are given by

$$s \in T \overset{\text{def}}{\iff} \mathcal{A} \text{ accepts } s,$$
$$a \in A \overset{\text{def}}{\iff} \text{trace}_{\mathcal{A}}(a) \in \Omega.$$

Lemma 2.11. *For any strategy $\sigma \subseteq {}^{<\omega}Q$ on the tree of traces, a strategy $\tau \subseteq {}^{<\omega}\Sigma$ on the tree of moves exists such that $\{\text{trace}(t)\ ;\ t \in \tau\} = \sigma$.*

[Obvious, by choosing a single move to represent each edge in the graph of the automaton.]

Definition 2.12 (Automata strategies). Let $\mathcal{A} = \langle Q, q_0, Q_{\mathbf{I}}, Q_{\mathbf{II}}, \delta\rangle$ be an arbitrary alternating automaton with alphabet Σ. An AUTOMATON STRATEGY for a game played on $\mathcal{A}$ is a structure

$$\langle M, m_0, \sigma\rangle$$

where M is an arbitrary set known as the MEMORY, containing at least one element $m_0 \in M$, the STARTING CONFIGURATION, and $\sigma\colon Q \times M \to \Sigma \times M$ is a function that both chooses a move and updates the memory. An automaton strategy for **I** is defined on all pairs in $Q_{\mathbf{I}} \times M$, and likewise for **II** on $Q_{\mathbf{II}} \times M$. We refer to the strategy as a whole as "σ" where this will not cause confusion.

While the plays in an automaton game are just ω-words, to match them to automaton strategies we have to go via the states of the automaton.

Definition 2.13 (Plays according to σ). Let $\mathcal{A} = \langle Q, q_0, Q_{\mathbf{I}}, Q_{\mathbf{II}}, \delta\rangle$ be an alternating automaton with alphabet Σ, $\langle M, m_0, \sigma\rangle$ a strategy for **I** (the alterations necessary for **II** are obvious). Let $a_{\mathbf{II}} \in {}^{\omega}\Sigma$ be an arbitrary ω-word, representing the moves by **II**.

The definition of the play constructed by σ is given inductively in terms of $a_{\mathbf{I}} \in {}^{\omega}\Sigma$ (the moves played by **I**, generated by σ), $\vec{q} \in {}^{\omega}Q$ (the states passed through by $\mathcal{A}$), $\vec{m} \in {}^{\omega}M$ (the memory states used by σ) and $a_{\mathbf{II}}$:

$$\begin{aligned}\vec{q}(0) &= q_0,\\ \vec{m}(0) &= m_0,\end{aligned}$$

For $n = 2k$ (moves by **I**):

$$\begin{aligned}\langle a_{\mathbf{I}}(k), \vec{m}(k+1)\rangle &= \sigma(\vec{q}(2k), \vec{m}(k)),\\ \vec{q}(2k+1) &= \delta(\vec{q}(2k), a_{\mathbf{I}}(k));\end{aligned}$$

For $n = 2k+1$ (moves by **II**):

$$\vec{q}(2k+2) = \delta(\vec{q}(2k+1), a_{\mathbf{II}}(k)).$$

This should be made clearer by a picture:

δ	$\vec{q}$	σ	$a_{\mathbf{I}}$	$\vec{m}$	$a_{\mathbf{II}}$
	q_0			m_0	
		$\sigma(q_0, m_0) =$	$\langle a_0,$	$m_1\rangle$	
$\delta(q_0, a_0) =$	q_1				a_1
$\delta(q_1, a_1) =$	q_2	$\sigma(q_2, m_1) =$	$\langle a_2,$	$m_2\rangle$	
$\delta(q_2, a_2) =$	q_3				a_3
$\delta(q_3, a_3) =$	q_4	$\sigma(q_4, m_2) =$	$\langle a_4,$	$m_3\rangle$	
$\delta(q_4, a_4) =$	q_5				a_5
	$\vdots$		$\vdots$	$\vdots$	$\vdots$

Finally, we construct the resulting play $a \subseteq {}^{\omega}\Sigma$ as follows: for $k \in \omega$,

$$a(2k) = a_{\mathbf{I}}(k),$$
$$a(2k+1) = a_{\mathbf{II}}(k).$$

We call a the RESPONSE of σ and $\mathcal{A}$ to $a_{\mathbf{II}}$. A play $b \in {}^{\omega}\Sigma$ is ACCORDING TO σ and $\mathcal{A}$ if there is some $c \in {}^{\omega}\Sigma$ such that b is the response of σ and $\mathcal{A}$ to c. The sequence $\vec{q}$ is the STATE RESPONSE of σ and $\mathcal{A}$ to $a_{\mathbf{II}}$, and such a state sequence is also ACCORDING TO σ and $\mathcal{A}$ if a suitable $c \in {}^{\omega}\Sigma$ exists.

We need to update our definition of strategy equivalence: two strategies (automata or ordinary tree-form) σ and τ are PLAY EQUIVALENT if the plays according to σ are exactly the plays according to τ.

The formal details of this definition are tiresome, but the idea is fairly simple. An automaton strategy isolates the computational mechanism used to produce the next move, and the size of the memory M provides a measure of the complexity of that mechanism. The terminology for automaton strategies is deliberately reminiscent of that for ordinary tree-form strategies. The following two theorems justify this notational overloading.

Theorem 2.14. *Let σ be an automaton strategy for the automaton game $\langle \mathcal{A}, \Omega \rangle$ with state set Q. Then there exists a strategy $\tau \subseteq {}^{<\omega}Q$ such that $\vec{q} \in [\tau]$ iff $\vec{q}$ is a state sequence according to σ and $\mathcal{A}$.*

In ordinary words: every automaton strategy can be transformed into an equivalent (up to traces) strategy on the trace tree.

Proof. Obvious, by an inductive construction taking partial responses of σ to finite sequences. q.e.d

Theorem 2.15. *Let $\sigma \subseteq {}^{<\omega}Q$ be a strategy (on the trace tree) for an automaton game $\langle \langle Q, q_0, \delta \rangle, \Omega \rangle$. Then there exists an automaton strategy $\langle M, m_0, \tau \rangle$ play equivalent to σ.*

Proof. Take $M = {}^{<\omega}Q$, $m_0 = \langle\rangle$, and define τ as follows:

$$\tau(q_0, \langle\rangle) = \langle \sigma(\langle\rangle), \delta(q_0, \sigma(\langle\rangle)) \rangle,$$

and for $s \neq \langle\rangle$,

$$\tau(q, s) = \left\langle \sigma(s^\frown q), s^\frown q^\frown \delta(q, \sigma(s^\frown q)) \right\rangle$$

While this definition looks complicated, it simply ensures that the memory always contains the previous history of the play so far. That is, repeating the same picture as before:

δ	$\vec{q}$	τ	$a_{\mathbf{I}}$	$\vec{m}$	$a_{\mathbf{II}}$
	q_0			$\langle\rangle$	
		$\tau(q_0, \langle\rangle) =$	$\langle a_0,$	$\langle q_1 \rangle\rangle$	
$\delta(q_0, a_0) =$	q_1				a_1
$\delta(q_1, a_1) =$	q_2	$\tau(q_2, \langle q_1 \rangle) =$	$\langle a_2,$	$\langle q_1, q_2, q_3 \rangle\rangle$	
$\delta(q_2, a_2) =$	q_3				a_3
$\delta(q_3, a_3) =$	q_4	$\tau(q_4, \langle q_1, q_2, q_3 \rangle) =$	$\langle a_4,$	$\langle q_1, q_2, q_3, q_4, q_5 \rangle\rangle$	
$\delta(q_4, a_4) =$	q_5				a_5
	$\vdots$		$\vdots$	$\vdots$	$\vdots$

Clearly, the memory is tracking the play history. Since the move chosen by τ is simply the response of σ to that history, the response to any $a_{\mathbf{II}}$ is a branch of σ. q.e.d

So we can freely transform back and forth between ordinary strategies (on traces) and automata strategies. But in the general case, the automaton strategy corresponding to a trace strategy requires infinite memory. We are particularly interested in cases where not all of the play history need be used to produce a winning strategy.

Definition 2.16. Let $\langle M, m_0, \sigma \rangle$ be an automaton strategy. If $|M| = 1$, we call σ MEMORYLESS. If $|M| < \omega$, σ is FINITE-MEMORY.

Lemma 2.17. *If the plays according to a function $\sigma\colon Q \to \Sigma$ are defined for each player in the obvious manner, then for every memoryless automaton strategy $\langle M, m_0, \tau \rangle$ there exists a function $\rho\colon Q \to \Sigma$ which is play equivalent to τ.*

[Obvious.]

By an abuse of terminology we refer also to ρ in the above lemma as a memoryless strategy. Our first example of such a strategy is for the reachability game, however it will be useful to study instead a generalised version based on the combinatorial game.

2.3.1 The combinatorial automaton game

The combinatorial automaton game is a generalised form of the reachability game. The first player unable to move loses, while infinite plays are a draw. In fact we have already seen this game, in section 1.6: the combinatorial automaton game on an automaton $\mathcal{A}$ is simply the combinatorial game on the unravelled tree of the graph of $\mathcal{A}$.

Definition 2.18. If $\mathcal{A} = \langle Q, q_0, Q_{\mathbf{I}}, Q_{\mathbf{II}}, \delta \rangle$ is an alternating automaton, the COMBINATORIAL AUTOMATON GAME on $\mathcal{A}$ is a pair

$$\langle \langle Q \cup \{\bot_{\mathbf{I}}, \bot_{\mathbf{II}}\}, q_0, Q_{\mathbf{I}} \cup \{\bot_{\mathbf{I}}\}, Q_{\mathbf{II}} \cup \{\bot_{\mathbf{II}}\}, \delta' \rangle, \Omega_{\mathrm{C}} \rangle$$

where δ' is given by

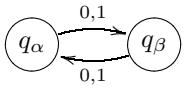
$$\delta'(q,x) = \begin{cases} \delta(q,x) & \text{if this is defined, otherwise} \\ \bot_{\mathbf{I}} & \text{if } q \in Q_{\mathbf{I}} \cup \{\bot_{\mathbf{I}}\}, \\ \bot_{\mathbf{II}} & \text{if } q \in Q_{\mathbf{II}} \cup \{\bot_{\mathbf{II}}\}. \end{cases}$$

and **I** wins traces containing $\bot_{\mathbf{II}}$, **II** wins traces containing $\bot_{\mathbf{I}}$, and all other traces are a draw.

Remark. The definition of the combinatorial automaton game cannot be given in the formalism we use for other automata games, because specifying only the winning set for **I** does not fully describe the payoffs. This is no great problem, since we will be using the game only to assist with various proofs, and these will rely on results shown on the combinatorial game on trees, for which we *do* have a proper formal representation. Likewise, the derived automaton $\mathcal{A}'$ is not properly speaking an alternating automaton, since both players play from the states $\bot_{\mathbf{I}}$ and $\bot_{\mathbf{II}}$. This could easily be remedied, but at the cost of some clarity. For similar reasons, we cannot strictly speaking place the combinatorial automaton game in the Borel hierarchy. However 'morally speaking' it is $\mathbf{\Sigma}^0_1$, since the sets of winning plays for both players are open.

Theorem 2.19. *For every reachability game $\langle \mathcal{A} = \langle Q, q_0, Q_{\boldsymbol{I}}, Q_{\boldsymbol{II}}, \delta\rangle, \Omega_{\mathrm{R}}\rangle$ generated by $Q_{\mathrm{T}} \subseteq Q$, there exists a combinatorial automaton game on an alternating automaton $\mathcal{B}$ with the same set of winning traces.*

Proof. Form the new alternating automaton $\mathcal{B}$ by making the following changes to $\mathcal{A}$: remove all transitions $\langle q, x, p\rangle \in \delta$ where $q \in Q_{\mathrm{T}}$. Add a new state $q_\bot$, and transitions $\delta(q,x) = q_\bot$ for all $x \in X$ and $q \in Q_{\mathbf{I}} \cap Q_{\mathrm{T}}$. Also add the system



and for every $x \in X$, for each $q \in Q_{\mathbf{I}} \setminus Q_{\mathrm{T}}$ a transition $\delta(q,x) = q_\alpha$, and for each $q \in Q_{\mathbf{II}} \setminus Q_{\mathrm{T}}$, a transition $\delta(q,x) = q_\beta$.

This last step ensures that **II** never wins the combinatorial automaton game (**I** always has an option to draw). And a play wins for **I** only if it passes through an element of Q_{T}, since these are the only leaves in the graph of $\mathcal{B}$. q.e.d

2.4 Memory requirement as complexity

The results in this section show how the memory requirements for automata strategies compare to the set-theoretic complexity analysis in terms of the payoff set.

Theorem 2.20. *Memoryless strategies suffice for combinatorial graph games. That is, if a winning (resp. non-losing) strategy exists for a combinatorial game $\mathrm{G}(A,B)$, then a memoryless winning (resp. non-losing) automaton strategy exists.*

Proof. Recall that the combinatorial Gale-Stewart labelling ℓ_{CGS} (Definition 1.22) is $\mathrm{G}(A, B)$-sound where A and B are open (specifically, where $\mathrm{G}(A, B)$ is a combinatorial game, the rulification of a tree). The proof proceeds by defining a similar labelling on the graph (which produces a memoryless automaton strategy), and showing that the unravelling of the labelled graph is the labelled tree. Since the tree labelling is $\mathrm{G}(A, B)$-sound, so is the graph labelling.

Definition 2.21. The COMBINATORIAL GRAPH LABELLING PROCEDURE labels the graph $G = \langle Q, E\rangle$ of an alternating automaton $\langle Q, q_0, Q_{\mathbf{I}}, Q_{\mathbf{II}}, \delta\rangle$ as follows:

$$\ell_0(q) \stackrel{\text{def}}{=} \begin{cases} \mathbf{I}_0 & \text{if } \mathrm{succ}_G(q) = \varnothing \wedge q \in Q_{\mathbf{II}}, \\ \mathbf{II}_0 & \text{if } \mathrm{succ}_G(q) = \varnothing \wedge q \in Q_{\mathbf{I}}, \\ \bot & \text{otherwise;} \end{cases}$$

$$\ell_{n+1} \stackrel{\text{def}}{=} \mathrm{backup}(\ell_n);$$

$$\ell_\omega \stackrel{\text{def}}{=} \bigcup\{\ell_n \,;\, n \in \omega\};$$

$$\ell_{\mathrm{CG}}(q) \stackrel{\text{def}}{=} \begin{cases} \ell_\omega(q) & \text{if this is defined,} \\ \mathbf{D}_0 & \text{otherwise.} \end{cases}$$

Lemma 2.22. *For some $k \in \omega$, $\ell_k = \ell_\omega$.*

[Each stage is either a fixpoint or labels one new vertex. There are only finitely many vertices to label.]

Let ℓ_{CGS} be the labelling produced by the combinatorial Gale-Stewart procedure on the unravelled tree T of G, with the seeds $A = \{(s^\frown q) \in T \,;\, \mathrm{succ}_G(q) = \varnothing \wedge \mathrm{lh}(s) \text{ is even}\}$, $B = \{(s^\frown q) \in T \,;\, \mathrm{succ}_G(q) = \varnothing \wedge \mathrm{lh}(s) \text{ is odd}\}$. (We must also add $\langle\rangle$ to B if $\mathrm{succ}_G(q_0) = \varnothing$.)

Let ℓ_{CG} be the labelling produced by the combinatorial graph labelling procedure on G.

Let $\sigma\colon Q \to X$ be an ℓ_{CG}-good memoryless strategy for player i. Derive a strategy τ on the trace tree by taking, for $s \in {}^{<\omega}Q$ owned by i,

$$\mathrm{succ}_\tau(s) = \left\{s^\frown\sigma(s(\mathrm{lh}(s) - 1))\right\}$$

(and $\mathrm{succ}_\tau(\langle\rangle) = \langle\sigma(q_0)\rangle$ if σ is a strategy for $\mathbf{I}$).

Lemma 2.23. *The strategies σ and τ are play-equivalent.*

[Obvious.]

Lemma 2.24. *The strategy τ is $\mathrm{G}(A, B)$-sound.*

[The key insight here is that the procedure on the tree includes every step taken by the procedure on the graph. That is, if $q \in Q$ has a successor q' labelled for example $\mathbf{I}_n$ in the graph, then every $s^\frown q \in {}^{<\omega}Q$ has a successor $s^\frown q^\frown q'$ labelled $\mathbf{I}_n$ in the tree. Furthermore, while the procedure on the tree does add labels not present on the graph, it adds only *higher* labels (essentially those caused by loops) which do not effect the formation of strategies.]

Lemmas 2.23 and 2.24 together guarantee that σ is sound for the combinatorial graph game, proving the theorem. q.e.d

Corollary 2.25. *Memoryless strategies suffice for the reachability game.*

Remark. For the more complex automata games, we often stipulate that the graph of the automaton is pruned, so that only 'true' infinite plays need be considered. Theorem 2.20 in a sense justifies this decision: if the automaton is not pruned, we can augment the condition with a combinatorial automaton game, solve this to produce memoryless strategies for both players, and solve the original game restricted to the pruned graph whose vertices receive **D** labels in the combinatorial game.

Theorem 2.26. *Memoryless strategies suffice for Büchi games.*

Proof sketch. Let $G = \langle Q, E\rangle$ be the graph of an automaton. Take the Büchi game from $q_0 \in Q$ with target set $Q_\mathrm{T} \subseteq Q$. As before, we will define a labelling on the graph. The details of the correspondence to the trace tree are omitted.

First, for each $q \in Q_\mathrm{T}$ define a labelling ℓ_q for the reachability analysis with target set $\{q\}$. Recall that the reachability game from q_0 is not automatically won if $q_0 \in Q_\mathrm{T}$, so for each $q \in Q_\mathrm{T}$ such that $\ell_q(q) \in \mathrm{L}_\mathbf{I}$, **I** can force a cycle that includes q.

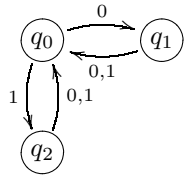
Let Q_C be the set of these 'cyclable' target states. Now use this as the seed for another reachability analysis. Call the resulting labelling ℓ_B.

That ℓ_B is sound also when lifted to the trace tree follows from the soundness of reachability labellings. For any $s \in {}^{<\omega}Q$ labelled $\mathbf{I}_n$ with $n > 0$, the soundness of reachability labellings guarantees that every $a \in [s]$ (restricted to an ℓ_B-good strategy) passes through a position labelled $\mathbf{I}_0$. Likewise, for every such position, it is guaranteed that every branch passes through another such, and the Büchi winning condition is satisfied by an inductive argument.

Conversely, from a position s labelled **II**, the branches through s do not pass through a vertex labelled $\mathbf{I}_0$. That this is sufficient for **II** to win is a consequence of the choice of Q_C; again by reachability, for $q \in Q_\mathrm{T} \setminus Q_\mathrm{C}$, **II** has a strategy that does not return to q. If $\ell_\mathrm{B}(s^\frown q) \in \mathrm{L}_\mathbf{II}$, and if $\tau \subseteq {}^{<\omega}Q$ is an ℓ_B-good strategy, then for all $a \in [\tau] \cap [s^\frown q]$, q does not recur in a, nor do any elements of Q_C. Furthermore, since the same holds for all $q \in Q_\mathrm{T} \setminus Q_\mathrm{C}$ labelled **II**, and since if $s \subseteq a$ is labelled **II** the same will hold for all $t \subseteq a$ (since τ is ℓ_B-good), no element of Q_T occurs infinitely often in a. q.e.d

Theorem 2.27. *Memoryless strategies do not suffice for Muller games. That is, there exists a Muller game for which **I** has a winning strategy but no winning memoryless strategy.*

Proof. Take the automaton



with the condition $\Omega_{\mathrm{M}} = \{\{q_1, q_2\}\}$. It is clear that **I** has a winning strategy (for instance, alternating playing 0 and 1) and equally clear that no winning memoryless strategy exists. q.e.d

So far it seems that increasing memory requirements correspond to increasing topological complexity. Our first example of a game that does not always admit memoryless strategies is also the highest in the Borel hierarchy. However the correspondence is only apparent. To show this, we need to introduce one more automaton condition.

Definition 2.28. The PARITY CONDITION is a function $f\colon Q \to \omega$ assigning to each state a value. We lift f to traces in the obvious manner: $\mathrm{trace}_f(s) = \langle f(\mathrm{response}(s{\restriction}n)) \ ; n \leq \mathrm{lh}(s)\rangle$. Player **I** wins a play a with parity condition f if the highest value occurring infinitely often in $\mathrm{trace}_f(a)$ is even.

Theorem 2.29 ([Far02, Theorem 1.22]). *The class of ω-languages accepted by parity automata is precisely those accepted by Muller automata.*

This of course means that the topological complexity of these two conditions is the same.

Theorem 2.30 ([Küs02]). *Memoryless strategies suffice for parity condition games.*

Theorems 2.29 and 2.30 demonstrate that the memory requirements complexity measure is sensitive to details of the games that do not affect the topology of the payoff sets. Less obviously, they place an upper bound on memory requirements for Muller games.

Theorem 2.31 ([Maz02, Theorem 2.7]). *Finite memory suffices for Muller games.*

Proof sketch. By Theorem 2.29, for every Muller condition automaton $\mathcal{A}$ there is a parity condition automaton $\mathcal{B}$ that accepts the same ω-language. Suppose **I** has a winning strategy for the game on $\mathcal{B}$. By Theorem 2.30, he has a memoryless winning strategy, say σ.

Now he can build a winning automaton strategy τ for $\mathcal{A}$, using the states of $\mathcal{B}$ for the memory. The memory traces a simulated path through $\mathcal{B}$, and σ generates his moves. Since σ is winning, the resulting play is accepted by $\mathcal{B}$, and therefore also by $\mathcal{A}$. q.e.d

2.5 Summary

The memory requirements measure of complexity does not correspond in any obvious manner to the topological complexity hierarchy. Instead, it is sensitive to the class of automaton game being played. While Muller games require more memory than the other games we considered (that lie lower in the Borel hierarchy and admit memoryless strategies), the class of Muller languages is the same as the class of parity languages, and parity automata admit memoryless strategies.

Chapter 3

Time complexity analysis

The last measure of game complexity considered in this thesis can be summarised as: "How difficult is it to *construct* a winning strategy?" This is the most concrete complexity measure we are concerned with; we take practical algorithms (operating thus only on finite structures) and measure their time complexity (the rate at which the running time of the algorithm increases with increasing problem size).

3.1 Definitions

Definition 3.1 (Function complexity). Let $f, g \colon \omega \to \omega$ be functions. The complexity class $\mathrm{O}(g)$ is the class of functions whose asymptotic growth is no faster than that of g.

We say $f \in \mathrm{O}(g)$ (in words, "f is ORDER g") iff there exist constants $a, b, c \in \omega$ such that for all $x > a$, $f(x) < bg(x) + c$. For convenience, we often give g directly as a polynomial:

- $f \in \mathrm{O}(1)$ ("f is CONSTANT TIME") for $f \in \mathrm{O}(n \mapsto 1)$;
- $f \in \mathrm{O}(n)$ ("f is LINEAR TIME") for $f \in \mathrm{O}(n \mapsto n)$;
- $f \in \mathrm{O}(n^2)$ ("f is QUADRATIC TIME") for $f \in \mathrm{O}(n \mapsto n^2)$, etc.

For algorithmic time complexity, given an algorithm `A` we take f to be the function mapping the algorithm input onto the number of basic operations the algorithm performs before terminating. Thus the statement "`A` is $\mathrm{O}(n)$" means that the algorithm takes time proportional to the size of its input (up to an additive constant) to terminate.

Making this definition precise requires two additional elements: the basic operations of an algorithm need defining, and the size of a structure.

Definition 3.2 (Algorithmic time complexity). A BASIC OPERATION in an algorithm is one of the following operations:

- membership test: "$v \in V$" returning a truth value;
- empty test: "$v = \varnothing$" returning a truth value;
- boolean and numeric operations, numeric comparisons (eg., addition, conjunction, equality test);
- assignment to a variable: "`x` $\leftarrow 3$";
- returning a value from a procedure: "`return` v";
- *each* iteration step over a set: "`foreach` $v \in V$";
- procedure call: "`Label(G,`u`,1)`".

The last two cases require some elaboration: the intention is that complex operations such as calling procedures or iterating over sets have costs attached to the control structure as well as to the work done in the loop or procedure body. So iterating through a set of n elements takes n basic operations in addition to those used up in the loop body, and the procedure call itself is considered one basic operation.

The SIZE of a structure is defined recursively. The size of a set of natural numbers or a relation $R \subseteq \omega \times \omega$ is simply its cardinality. The size of a structure (a sequence of sets or structures) is the sum of the sizes of its components, plus its length. In particular, the size of a graph $G = \langle V, E \rangle$ is $|V| + |E| + 2$.

Let `A` be an algorithm taking structures from a set $\mathcal{S}$ as input. The ALGORITHMIC TIME COMPLEXITY of `A` is the complexity of the function $f : x \mapsto\!= t_x$, where t_x is given by

$$t_x = \max_{S \in \mathcal{S}} \{\text{basic operations of } \texttt{A} \text{ operating on } S \,;\, S \text{ is of size } x\}.$$

Remark. The algorithmic language used here is a simplistic idealisation of real programming languages. If this idealisation is not to give unrealistic results, we must ensure that the proposed basic operations can in fact be given constant-time implementations, and that these implementations then correspond to the input size measurement given above. This can be achieved using a neighbour-list implementation of graphs: each vertex stores a linked list of its successors and another of its predecessors.

This implementation gives linear iteration time over successors and predecessors in the graph, but makes updating the graph structure highly expensive; however all the algorithms given here are read-only on the graph. The measure of graph size given above is correct up to a (multiplicative) constant factor: each edge in fact contributes two linked list elements (one for a successor list and one for a predecessor list) but this constant factor is absorbed in the definition of function complexity.

It will be convenient to define a slightly different type of labelling to that given before. The labels are **W**, **D** and **L**, with intended meanings "the player

playing from this position has a winning/a drawing/no non-losing strategy", respectively. This notation makes the algorithms less verbose, since the backward induction step functions symmetrically for the two players.

Definition 3.3 (Labellings again). The sets

$$\begin{aligned} \mathrm{L}_{\mathbf{W}} &\stackrel{\text{def}}{=} \{\mathbf{W}_n \,;\, n \in \omega\}, \\ \mathrm{L}_{\mathbf{D}} &\stackrel{\text{def}}{=} \{\mathbf{D}_n \,;\, n \in \omega\}, \\ \mathrm{L}_{\mathbf{L}} &\stackrel{\text{def}}{=} \{\mathbf{L}_n \,;\, n \in \omega\} \end{aligned}$$

are called respectively the WIN LABELS, DRAW LABELS and LOSE LABELS.

A WIN/LOSE LABELLING on a bipartite graph $\langle V, E\rangle$ with $V_{\mathbf{I}} \subseteq V$ and $V_{\mathbf{II}} = V \setminus V_{\mathbf{I}}$ the vertices owned by **I** and **II** is a function

$$\ell \colon V \to \mathrm{L}_{\mathbf{W}} \cup \mathrm{L}_{\mathbf{D}} \cup \mathrm{L}_{\mathbf{L}}.$$

On bipartite graphs the two formalisms are interchangeable, according to the following INTERPRETATION BIJECTION $\mathcal{L}_{V_{\mathbf{I}}}$:

$$\begin{aligned} \langle v \in V_{\mathbf{I}}, \mathbf{W}_n\rangle &\mapsto \langle v, \mathbf{I}_n\rangle\,, \\ \langle v \in V_{\mathbf{II}}, \mathbf{W}_n\rangle &\mapsto \langle v, \mathbf{II}_n\rangle\,, \\ \langle v \in V_{\mathbf{I}}, \mathbf{L}_n\rangle &\mapsto \langle v, \mathbf{II}_n\rangle\,, \\ \langle v \in V_{\mathbf{II}}, \mathbf{L}_n\rangle &\mapsto \langle v, \mathbf{I}_n\rangle\,, \\ \langle v, \mathbf{D}_n\rangle &\mapsto \langle v, \mathbf{D}_n\rangle\,. \end{aligned}$$

We lift the notion of heights to win/lose labellings in the obvious way.

Remark. This notation can also be used for non-bipartite graphs, for which the algorithms presented in this section were originally developed, however for simplicity we stick to bipartite graphs. For all the games considered here, the extension to non-bipartite graphs is trivial in principle but for many the algorithms and definitions become much more convoluted.

3.2 Backward induction on the graph

The backward induction algorithm for a bipartite graph was given as a labelling procedure in Definition 2.21. Here I formalise this procedure and give an upper bound on its time complexity.

[Fra97] gives an informal algorithm for backward induction on a non-bipartite graph, labelling the vertices **W**, **L** and **D**.[1] This algorithm is stated to be $\mathrm{O}(|E|)$ (where E is the edge set for the graph) however this statement is misleading. Some of the operations apparently[2] taken as primitive involve operations on sets of vertices (such as collecting all those with no successors) which if taken separately would add inner loops to the algorithm.

[1] The notation in the paper is different: "N" and "P" replace **W** and **L**, respectively.
[2] No formal analysis of the time complexity of this algorithm is given in the paper.

To properly consider the question, Figure 3.1 contains a formal version of the algorithm given in [Fra97]. This is the combinatorial graph labelling procedure (Definition 2.21) and is in fact also suitable for alternating games on non-bipartite graphs. I call it the "naive" backward induction algorithm, because the running time can in fact be improved with some additional bookkeeping.

Definition 3.4. Let $\langle V, E\rangle$ be a bipartite graph with player-owned sets $V_{\mathbf{I}} \subseteq V$ and $V_{\mathbf{II}} = V \setminus V_{\mathbf{I}}$. Then $\ell_{\texttt{CG-n}}$ is the labelling produced by `CG-naive`, interpreted according to the bijection $\mathcal{L}_{V_{\mathbf{I}}}$.

Theorem 3.5. *The labelling $\ell_{\texttt{CG-n}}$ is sound for the combinatorial graph game.*

Proof. The proof is by showing the equivalence of `CG-naive` to the combinatorial graph labelling procedure. This equivalence is not strict (i.e., in general $\ell_{\texttt{CG-n}} \neq \ell_{\mathrm{CG}}$) but they are equivalent up to the height of vertices.

Firstly, it is obvious by inspection of the procedure `Label` that all labels are applied correctly. (That is, if `Label` labels a vertex **W** or **L** it does so according to the same rules as the inductive step constructing ℓ_{CG}.) It remains to show that every vertex that ℓ_{CG} labels **W** or **L** also gets labelled by the algorithm (i.e., that **D** labels are only applied where correct).

```
Procedure:MAIN(G = ⟨V,E⟩:graph)
foreach v ∈ V do
    if (succ(v) = ∅) then
        Ell[v] ← L_0;
        foreach {u ∈ pred(i) ; Ell[u] = undef} do
            Label(G,u,1);
foreach v ∈ V do
    if Ell[v] = undef then
        Ell[v] ← D_0;

Procedure:Label(G = ⟨V,E⟩:graph; v ∈ V:vertex, h ∈ ω)
aux ← L_h;
foreach u ∈ succ(v) do
    if (Ell[u] ∈ L_L) then
        aux ← W_h;
    else if (Ell[u] = undef and aux = W_h) then
        aux ← undef;
if (Ell[v] ≠ aux) then
    Ell[v] ← aux;
    foreach {u ∈ pred(v) ; Ell[u] = undef} do
        Label(G,u,h+1);
```

Figure 3.1: `CG-naive`: Naive backward induction on the graph

The proof is by induction on the height of vertices, with the base case that all vertices of height 0 are labelled eventually.

For the inductive step, assume that all vertices of height less than n in ℓ_{CG} get labelled at some point by `CG-naive`. Let v be a vertex of height n in the abstract procedure, and we show that it also gets labelled by `CG-naive`.

First, suppose v is labelled $\mathbf{W}_n$ by ℓ_{CG}. Then some successor u of v was labelled $\mathbf{L}_m$, with $m < n$. By the inductive hypothesis, u gets labelled by `CG-naive`, and that labelling triggers the call to `Label` that will label v. (It is possible that v has *already* been labelled when this occurs; it is this condition that produces the discrepancy between the heights according to the two labellings, however it does not affect the correctness.)

Now suppose instead that v is labelled $\mathbf{L}_n$ by ℓ_{CG}. In this case, all successors of v must have been labelled at earlier stages. Similarly, the inductive hypothesis guarantees that they are at some point labelled by `CG-naive`, and the last such labelling triggers the call to `Label` that will label v.

Since all $\mathbf{W}$ and $\mathbf{L}$ labels are correct, and all vertices labelled $\mathbf{W}$ and $\mathbf{L}$ by ℓ_{CG} eventually get labelled (and the $\mathbf{D}$ labels are obviously correct, by inspection of the loop), the labels applied by `CG-naive` are equivalent up to height to those of ℓ_{CG}.

Thus, since ℓ_{CG} is sound for the combinatorial graph game, so is $\ell_{\texttt{CG-n}}$. q.e.d

Theorem 3.6. `CG-naive` *is* $\mathrm{O}(|E| \cdot |V|)$ *but not* $\mathrm{O}(|E|)$.

Proof. The sub-procedure `Label` gets called at most once for every edge in the graph (the `foreach` loops containing `Label` calls). The loop through successors multiplies this factor by (in the worst case) the number of vertices. Since all other operations are constant-time, `CG-naive` is $\mathrm{O}(|E| \cdot |V|)$.

To see that $\mathrm{O}(|E|)$ is too strict an upper bound, consider the family of graphs given in Figure 3.2.

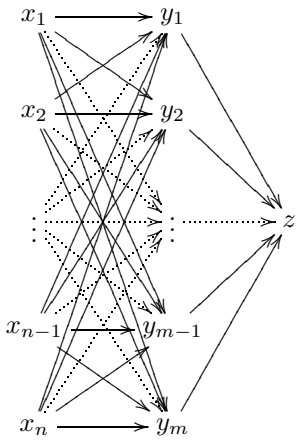


Figure 3.2: A family of graphs with poor labelling properties

The vertices y_i will all receive the label **W**. Each time this happens, all the vertices x_j will be passed to `Label` (that is, once for every edge between x_j and y_i), and the successor check will loop through every vertex y_k. This gives a total running time of at least $O(nm^2)$ — the first nm is $O(|E|)$, the number of edges between vertices x_i and y_j, while m is of course $O(|V|)$. q.e.d

However, we can do better. With a sacrifice to the readability of the algorithm, we can reduce the running time to a real $O(|E|)$. The algorithm is given in Figure 3.3, and uses a count of winning successors for each vertex to avoid repeatedly querying successors.

```
   Procedure:MAIN(G = ⟨V, E⟩:graph)
   foreach v ∈ V do
    succ_count_v ← 0;
   foreach ⟨u, v⟩ ∈ E do
    succ_count_u ← succ_count_u + 1;
   foreach v ∈ V do
      if (succ_count_v = 0) then
       Label(G, v, 0);

   foreach v ∈ V do
      if Ell[v] = undef then
       Ell[v] ← D_0;

   Procedure:Label(G = ⟨V, E⟩:graph; v ∈ V:vertex, h ∈ ω)
   if Ell[v] ≠ undef then
    return;
   else if succ_count_v = 0 then
      Ell[v] ← L_h;
      foreach u ∈ pred(v) do
       Label(G, u, h + 1);
   else
      Ell[v] ← W_h;
      foreach u ∈ pred(v) do
(*)    succ_count_u ← succ_count_u − 1;
      foreach u ∈ pred(v) do
         if succ_count_u = 0 then
          Label(G, u, h + 1);
```

Figure 3.3: `CG-book`: backward induction with bookkeeping

Again, we must first prove that the resulting labelling is sound, then show the running time to be $O(|E|)$.

Theorem 3.7. `CG-book` *produces a sound labelling for the combinatorial game on* G.

Proof. We reduce this to the previous case by noting the following things:

1. The variables $\texttt{succ\_count}_v$ accurately reflect the number of successors of v not labelled **W** at each point in the algorithm at which they are queried;
2. The sub-procedure `Label` is called on a vertex v when either a successor u of v receives the label **L** (as in `CG-naive`) or when v has no successors not labelled **W** (the condition for applying the label **L** in `CG-naive`);
3. In both these cases the procedure performs the same labelling as `CG-naive`.

Taking these together, we can see that the same labellings are performed by both procedures, as required. q.e.d

Theorem 3.8. `CG-book` *is* $\mathrm{O}(|E|)$.

Proof. The `succ_count` update (line marked $(*)$) gets performed no more than once for each edge in G. Likewise the procedure `Label` is called no more than once for each edge.

(Note that every edge *is* inspected, for the initialisation of the `succ_count` variables, so we cannot do any better than this.) q.e.d

3.3 Asymmetric combinatorial game

The ASYMMETRIC COMBINATORIAL GRAPH GAME was the subject of [dJLar]. In this variant of the combinatorial graph game, **I** wins plays in which **II** exits the tree as usual, but **II** wins the infinite plays, and those in which **I** exits the tree are drawn.

The interesting feature of the labelling procedure we developed to solve this game is that it is nonmonotone: the labels applied to vertices at successive stages of the algorithm may change. Despite this, we showed that the algorithm is guaranteed to terminate in finite time, and with similar bookkeeping techniques to `CG-book` it is also $\mathrm{O}(|E|)$.

An alternative to this approach is provided by the labelling ℓ_{twice} (Definition 1.24). It would suffice to label the graph first for the game G_1 in which **I** wins if **II** exits the graph, and **II** wins otherwise, and then again for the game G_2 in which **I** wins any play that exits the graph. The existence of winning strategies for these two games is then translated back into the existence of strategies in the asymmetric combinatorial game via the preference ordering on outcomes: if **I** has a winning strategy in G_2 but not in G_1, for instance, then he has a drawing strategy in the asymmetric game.

Since a constant factor can be ignored for algorithmic time complexity, the asymmetric combinatorial game can be solved as efficiently as the ordinary combinatorial game, or the reachability game. [Lö3] considers the problem of games with more than two players, and finds that a preference order on outcomes and a set of rationality assumptions produce determinacy in a wide range of situations. In particular, the standard combinatorial graph game is a degenerate

case of an EXCEPTIONAL LEAST EVIL situation, in which a closed outcome **D** is preferred by all players over all outcomes except their most preferred outcome (i.e., in this case, both players would rather draw than lose). It would seem very easy to adapt the two-pass labelling procedure for more players, by introducing extra passes and interpreting the results in terms of the preference orderings of the players.

3.4 Higher complexity games

In the rest of this section we will give results about solution algorithms for some of the more complex games from automata theory. Given that we know we can deal in linear time with the combinatorial-game section of an automaton game (the vertices from which one player has a strategy forcing a play to a leaf) we now restrict our attention to pruned graphs. (Recall also that combinatorial games admit memoryless strategies, and that the payoff sets are open. Our three complexity measures agree that these games are in some sense 'easy'.)

The procedure given for the reachability game can be written as an algorithm in much the same way as `CG-naive`, and optimised using bookkeeping techniques like those of `CG-book`. Figure 3.4 shows the algorithm.

Definition 3.9. Call the labelling produced by `Reach` $\ell_{\mathtt{R}}$.

Theorem 3.10. *The labelling $\ell_{\mathtt{R}}$ is sound for the reachability game. (Recall that this game is not won automatically by starting at a vertex in V_{T}, but only by a* non-empty *walk ending in a target vertex.)*

Proof. An easy adaptation of the techniques used to prove soundness of $\ell_{\mathtt{CG}}$ and $\ell_{\mathtt{CG\text{-}n}}$. q.e.d

Theorem 3.11. `Reach` *is* $\mathrm{O}(|E|)$.

Proof. As before, `Label` gets called no more than once per edge, and likewise the `succ_count` update occurs no more than once per edge. q.e.d

We can use this algorithm (which is basically a trivial modification of `CG-book`) to solve Büchi games. The algorithm given in Figure 3.5 finds those vertices within the target set from which he can force a cycle passing through the target set. The full algorithm is given here for time complexity analysis purposes, but it is more clearly expressed inductively.

Definition 3.12. The procedure Cycle is defined as follows:

$$V_0 \stackrel{\text{def}}{=} V_{\mathrm{T}},$$
$$V_{n+1} \stackrel{\text{def}}{=} V_n \cap \mathrm{Reach}(V_n).$$

Lemma 3.13. Cycle *reaches a fixpoint after finite steps.*

[For each n, $V_{n+1} \subseteq V_n$, and V_{T} is finite.]

```
Procedure:MAIN(G = ⟨V, V_I, V_II, E⟩:bipartite_graph, V_T)
foreach v ∈ V do
    succ_count_u ← 0;
foreach ⟨u, v⟩ ∈ E do
    succ_count_u ← succ_count_u + 1;
foreach v ∈ V_T do
    succ_count_v = 0;
foreach v ∈ V_T do
    foreach u ∈ pred(v) do
        succ_count_u ← succ_count_u − 1;
    foreach u ∈ pred(v) do
        Label(G, u, 0);
foreach v ∈ V do
    if Ell[v] = undef then
        Ell[v] ← II_0;

Procedure:Label(G = ⟨V, V_I, V_II, E⟩:graph; v ∈ V:vertex, h ∈ ω)
if Ell[v] ≠ undef then
    return;
else if v ∈ V_I ∨ (v ∈ V_II ∧ succ_count_v = 0) then
    Ell[v] ← I_h;
    foreach u ∈ pred(v) do
        succ_count_u ← succ_count_u − 1;
    foreach u ∈ pred(v) do
        Label(G, u, h + 1);
```

Figure 3.4: `Reach`: reachability (with bookkeeping)

```
Procedure:MAIN(G = ⟨V, V_I, V_II, E⟩:bipartite_graph, V_T)
  returns V' ⊆ V_T
V' ← Catch(G,V_T);
while V' ≠ V_T do
  V_T ← V';
  V' ← Catch(G,V_T);
return V';
Procedure:Catch(G = ⟨V, V_I, V_II, E⟩:bipartite_graph, V_T)
  returns V' ⊆ V_T
foreach v ∈ V do
  Ell[v] ← undef;
Reach(G,V_T);
V' ← ∅;
foreach v ∈ V_T do
  if Ell[v] = I then
    V' ← V' ∪ {v};
return V';
```

Figure 3.5: `Cycle`: find vertices from which **I** can force a cycle

Lemma 3.14. *The fixpoint appears after no more than* $|V_{\mathrm{T}}|$ *steps.*

[Each non-fixpoint stage must remove at least one vertex from the set.]

Theorem 3.15. *Let* n *be the fixpoint of* Cycle. *Then for each* $v \in V_n$ *there exists* $u \in V_n$ *such that* ***I*** *has a winning strategy for the reachability games from* v *to* u *and from* u *to* u.

Proof. The proof is by induction on the subsequent stages of the algorithm after the fixpoint has been reached.

Let $v \in V_n$ be arbitrary, where n is the fixpoint. Since $v \in V_{n+1}$, for some $u \in V_n$ **I** has a winning strategy for the reachability game from v to u. If $u = v$ then we are done, otherwise apply the same reasoning for u and V_{n+2}: there must be some u' such that **I** wins the reachability game from u to u'.

Now note that winning strategies compose, so **I** also wins the game from v to u'. If $u = u'$ or $v = u'$ we are done, otherwise repeat for V_{n+3}. Since there are only finitely many vertices, we are guaranteed to find a cycle. q.e.d

Theorem 3.16. *The algorithm* `Cycle` *is* $\mathrm{O}(|E| \cdot |V|)$.

Proof. `Reach` is $\mathrm{O}(|E|)$. We must run `Reach` at most once for every vertex in V_{T}, which is bounded by $|V|$. q.e.d

Theorems 3.15 and 3.16 give an efficient solution for Büchi games. Sadly, this seems to be as far as we can get with efficient solution algorithms. In the

final days of preparation of this thesis, I discovered the following theorem, which will be presented in Gdansk the day after my thesis defence:

Theorem 3.17 ([HD05]). *Deciding the winner of a Muller game is* PSPACE*-complete.*

3.5 Summary

Like the automata theoretic analysis by memory requirements, the time complexity analysis is sensitive to details of the game construction beyond simply the topology of the payoff set. Perhaps more surprisingly, the hierarchy of games seems more closely allied to the Borel hierarchy than the muddle produced by the memory requirements. At least, by the techniques shown here, reachability games can be solved more easily than Büchi games, and the result of [HD05] places Muller games firmly above Büchi games.

Chapter 4

Software for graph algorithms

This thesis includes a piece of software written to experiment with graph games and labelling algorithms. There are two parts to the software: a core library of extendable classes defining graphs, labellings, and some aspects of algorithms, and a graphical application that makes use of the library to provide a 'playground' for experimentation.

4.1 The library

The library provides an abstract definition of graphs and sets, and a single implementation. In addition, it contains a number of classes for the Java event handling model, making it easy to build components that react to changes in a graph (vertices being added or removed, for instance) in a cleanly structured manner.

4.2 The application

The aim of the application is to let the user play with graphs and labelling algorithms. You can create and manipulate a graph with the mouse, and colour the vertices by hand. You can also apply simple labelling algorithms (reachability, backward induction for the combinatorial game) either in 'slow motion' to observe the individual steps, or simply for the results. The software is available for download as it was at the time the thesis was completed;[1] bug-fixes and updates will be posted on my homepage.[2]

[1] `http://www.illc.uva.nl/Publications/ResearchReports/MoL-2005-07.software.jar`
[2] `http://tikitu.dejager.net.nz/software/GameGraph/`

4.3 User manual

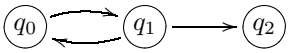
This section gives two views of the functioning of the software; the first explains what each user interface element (menu item, button, etc.) is for. After this, I give a task-oriented view ("How do I . . . ?").

4.3.1 What all the twiddly bits do

The **File** menu allows you to **Save** and **Open** graph description files. The format is intended to be human-readable and easily edited by hand.

Graph file format

Rather than give the full specification of the file format, I give here the file corresponding to the graph of the automaton shown in Figure 2.1, here reproduced for convenience:[3]




```
graph { 0..2 |
   0 --> 1;
   1 --> 0;
   1 --> 2;
}
layout {
   0: (50,50);
   1: (150,50);
   2: (250,50);
}
```

The `layout` section is optional, and if given specifies the position in pixel coordinates of the centre of a vertex (using the standard Java Swing model). Any vertices not positioned are placed at the top-left corner.

The first line of the `graph` declaration (between "{" and "|") declares the vertices that are to feature in the graph. This is a comma-separated list of single vertices and ranges, so the same vertices would be given by `0,1..2` or `0,1,2`. They must, however, be given in ascending order. It is an error to reference a vertex that has not been declared, either in the edges or in the `layout`.

Should it be necessary, comments can be introduced into graph files using "`//`" (all text between this comment symbol and the end of line will be discarded).

The Sets window

The **Windows** menu allows you to reveal the Sets window. Vertices can be placed into sets, and their membership displayed by changing their colours (ver-

[3] The software deals only with graphs, not automata, so the edges are not labelled.

tices belonging to more than one set will be coloured only for one of them). This mechanism is what we use for labelling.

The Sets window displays the special "Selection" set, and any others the user has created (or belonging to a currently running algorithm). You can add vertices to a set by selecting them (see next section) then using the **Copy Into . . .** control on the "Selection" set.[4]

Editing graphs

The **Edit Modes** menu lists the various controls for creating and editing new graphs (also provided as buttons on the toolbar). These modes alter the result of mouse clicks on the graph. In **Add/Remove** mode, clicking on a vertex removes it while clicking on empty space adds a new vertex there. **Drag** mode allows you to move existing vertices around. Clicking a vertex in **Select** mode selects it, while Control-clicking adds the vertex to the "Selection" set, letting you select multiple vertices at once. This can be useful for creating large graphs: the **Link multiple** mode adds edges from all selected vertices to the one you click on. (If this is unclear, try selecting several vertices then changing to **Link multiple** mode. Now hover the mouse over another vertex, and you will see the edges that will be added by clicking showing up in a lighter gray.)

Algorithms

There are two algorithms you can run, both accessible from the **Algorithms** menu. After selecting one, the buttons labelled "`<<`" (reset) and "`>`" (step) on the toolbar can be used to step through the algorithm or to reset its state. The labellings are indicated by sets `W`, `L` and `D`, coloured respectively blue, red and green, which can be viewed using the **Show Sets** item in the **Windows** menu.

The **Gale-Stewart** algorithm applies exactly the procedure given in Definition 1.13. That is, each step applies Backup to the labelling produced by the previous step, until this process reaches a fixpoint, then all unlabelled vertices are labelled `D`.

The **Backup-N** algorithm is an implementation of `CG-naive` (Figure 3.1) which inspects less vertices than the pure Gale-Stewart procedure (each vertex v must be inspected once, but is inspected a second time only if some successor of v becomes labelled). To indicate this, the vertex currently being inspected is shown by the set "current".

4.3.2 Task-oriented manual: How do I . . .

Each entry in this section answers a different "How do I . . . " question.

[4] The user interface in this window is not complete. There is no easy way to remove vertices from a set, and the mechanism for adding them is highly unsatisfactory.

. . . create graphs?

Either by hand-editing a file and using **Load** (in the **File** menu), or using the **Edit Modes**, as described in the next few entries.

. . . add vertices?

Use the **Add/Remove** edit mode, and click on an empty space.[5]

. . . remove vertices?

Use **Add/Remove** and click the vertex.

. . . add edges?

Use **Select** to select the source(s) of your edges. (If your graph has a dense and regular structure, you can save yourself some effort by choosing your selection carefully.) Now change to the **Link multiple** mode, and click on the destination(s) of your edges.

. . . remove edges?

Due to an oversight, this never made it into the graphical interface. You'll have to do it by saving to a file, editing the file, then loading it. Sorry.

. . . select vertices?

Use the **Select** mode. Clicking a vertex makes it the sole selection, clicking empty space empties the selection. Holding the Control (Ctrl) key and clicking adds or removes the vertex from the selection, which allows you to build up multiple-vertex selections.

. . . alter the layout?

Either by editing a file by hand (not recommended for layout) or using the **Drag** edit mode. (Note that this is independant of selection, so rearranging vertices will not change which ones you have selected.)

. . . label vertices?

First, make sure that the "Sets" window is showing (**Show Sets** in the **Window** menu).[6] A set can be used as a label. Use the **New Set** control to create a set (note that the algorithms use the sets `W`, `L`, `D` and `current` for special purposes, and that `Selection` is defined as the name of the set of selected vertices). Now

[5] The click detection requires that the mouse not move between pressing and releasing the button.

[6] If the checkbox is checked but the window is not visible, uncheck it then check it again. This can happen if the window is dismissed without using the menu control.

select the vertices you want to label, and use the **Copy Into** control on the "Selection" set.[7]

. . . run algorithms?

The two supported algorithms are described in section 4.3.1 above. If you want different algorithms, you'll have to write them as code and recompile the whole project. It was originally intended to include an algorithm description "mini-language" in the software, but this proved too large an undertaking.

[7] At this point there is no way to remove individual vertices from a set.

Appendix A

Differences

In this appendix the three techniques displayed for measuring the complexity of games are applied to Difference Games, a generalisation of reachability games.

A.1 The difference hierarchy

The definitions given here are taken from [LS05, section 2.1]. Intuitively, a set A at level n of the difference hierachy can be presented as $A = A_0 \cup (A_2 \setminus A_1) \cup (A_4 \setminus A_3) \cdots \cup (A_n \setminus A_{n-1})$, with $A_n \subseteq A_m$ if $n < m$, and minor alterations depending on the parity of n. This intuitive picture is insufficent once we reach infinite ordinal levels, but should be kept in mind when reading the following definitions.

Definition A.1 (Hausdorff difference hierarchy). For a sequence $\vec{A} = \langle A_\gamma \,;\, \gamma < \alpha \rangle$, the HAUSDORFF DIFFERENCE $\mathrm{Diff}(\vec{A})$ is the set

$$\left\{ x \in \bigcup_{\gamma<\alpha} A_\gamma \,;\, \min\{\gamma \,;\, x \in A_\gamma\} \text{ is of different parity to } \alpha \right\}.$$

(Recall that an ordinal is odd if it is of the form $\lambda + 2n + 1$ for λ a limit ordinal and $n \in \omega$, or even if the form is $\lambda + 2n$.)

Given a Polish space ${}^\omega X$, the HAUSDORFF DIFFERENCE CLASSES are defined as follows: for $A \subseteq {}^\omega X$, $A \in \alpha\text{-}\mathbf{\Sigma}^0_1$ if there is an increasing sequence of open sets $\vec{A}$ of length α such that $A = \mathrm{Diff}(\vec{A})$.

If $A = \mathrm{Diff}(\vec{A})$ we call $\vec{A}$ a PRESENTATION of A. Although in general the presentation of a set need not be unique, for sets in the Hausdorff difference hierarchy we can define a CANONICAL PRESENTATION $\vec{C}$ as follows:

Let $A \in \alpha\text{-}\underset{\sim}{\mathbf{\Sigma}}^0_1$ with α even. Then define

$$C_0 \stackrel{\text{def}}{=} \bigcup \big\{ [s] \; ; \; [s] \subseteq A \big\},$$

$$C_\beta \stackrel{\text{def}}{=} \bigcup \left\{ [t] \; ; \; [t] \subseteq ({}^\omega X \setminus A) \cup \bigcup_{\gamma<\beta} C_\gamma \right\} \text{ for } \textbf{odd } \beta,$$

$$C_\beta \stackrel{\text{def}}{=} \bigcup \left\{ [s] \; ; \; [s] \subseteq A \cup \bigcup_{\gamma<\beta} C_\gamma \right\} \text{ for } \textbf{even } \beta.$$

Finally, $\vec{C} = \langle C_\beta \; ; \; \beta < \alpha \rangle$ is by inspection a presentation of A.

Theorem A.2 (Hausdorff-Kuratowski, [LS05, Theorem 2.1]).

$$\underset{\sim}{\mathbf{\Delta}}^0_2 = \bigcup_{\alpha<\omega_1} \alpha\text{-}\underset{\sim}{\mathbf{\Sigma}}^0_1.$$

As mentioned in Chapter 1, [LS05] uses this result to show that all $\underset{\sim}{\mathbf{\Delta}}^0_2$ games have combinatorial labellings. In the following two sections, we develope a natural analogue of the difference hierarchy for automata games, and show how complex its strategies and analysis algorithms are.

A.2 Memory requirements for difference games

The natural automaton equivalent to the difference hierarchy is restricted to finite levels. The idea here is similar to the children's game "good news/bad news": The good news is, you won sky-diving tickets! The bad news is, your parachute didn't open. The good news is, you spotted a haystack! The bad news is, there was a pitchfork in the haystack. The good news is, you missed the pitchfork! The bad news is, you missed the haystack. And so on.[1] The game is defined by a sequence of target sets. First **I** must reach an element of the first set, then **II** must reach an element of the second, and so on. The first player to fail to accomplish a stage loses.

Definition A.3 (Good news/bad news game). Let $\langle Q, q_0, \delta \rangle$ be an automaton. A GOOD NEWS/BAD NEWS GAME of DIFFICULTY n, for $n \in \omega$, is defined by a sequence $\vec{Q} \in {}^{<\omega}(\wp(Q))$ of length n, called the REQUIREMENTS, as follows:

Let $\vec{q}$ be the trace of a play. Define the NEWS of the trace, $\text{news}(\vec{q}) \in {}^{<\omega}\omega$:

$$\text{news}(\vec{q})(0) \stackrel{\text{def}}{=} \begin{cases} \min\{x \in \omega \; ; \; \vec{q}(x) \in \vec{Q}(0)\} & \text{if this is defined} \\ \bot & \text{otherwise,} \end{cases}$$

$$\text{news}(\vec{q})(n+1) \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } \text{news}(\vec{q})(n) = \bot, \\ \min\{x > \text{news}(\vec{q})(n) \; ; \; \vec{q}(x) \in \vec{Q}(n+1)\} & \\ & \text{if this is defined,} \\ \bot & \text{otherwise.} \end{cases}$$

[1] The exclamation marks are traditional.

(Note that by this definition $\mathrm{lh}(\mathrm{news}(\vec{q})) \leq \mathrm{lh}(\vec{Q})$.)

Finally, **I** wins $\vec{q}$ iff $\mathrm{lh}(\mathrm{news}(\vec{q}))$ is odd ('good news').

Theorem A.4. *Let $G(A)$ be a good news/bad news game on some (finite) alphabet Σ with requirements $\vec{Q}$, and payoff set $A \subseteq {}^{\omega}\Sigma$. Then*

$$A \in \mathrm{lh}(\vec{Q})\text{-}\underset{\sim}{\mathbf{\Sigma}}^0_1.$$

Proof. For simplicity, we consider only the case for requirements of even length (that is, even difference classes).

Define a sequence of sets $\vec{A} = \langle A_i \in {}^{\omega}\Sigma \,;\, i \leq n \rangle$ as follows:

$$a \in A_i \overset{\text{def}}{\iff} \mathrm{lh}(\mathrm{news}(\mathrm{trace}(a))) \geq \mathrm{lh}(\vec{Q}) - i.$$

Now take $\mathrm{Diff}(\vec{A})$. I claim that $A = \mathrm{Diff}(\vec{A})$.[2] Since $\mathrm{lh}(\vec{A}) = \mathrm{lh}(\vec{Q})$ and since each A_i is obviously open, this will prove $A \in \mathrm{lh}(\vec{Q})\text{-}\underset{\sim}{\mathbf{\Sigma}}^0_1$.

Let $a \in \mathrm{Diff}(\vec{A})$ be arbitrary. Then $\mathrm{lh}(\mathrm{news}(\mathrm{trace}(a)))$ is odd, by construction. By definition, **I** wins a. So $\mathrm{Diff}(\vec{A}) \subseteq A$.

Now take $a \in A$ arbitrary. By definition, $\mathrm{lh}(\mathrm{news}(\mathrm{trace}(a)))$ is odd, and by the definition of "news", less than $\mathrm{lh}(\vec{Q})$. Then a appears in A_i for all $i \geq \mathrm{lh}(\mathrm{news}(\mathrm{trace}(a)))$, and the least of these is odd. Therefore $a \in \mathrm{Diff}(\vec{A})$, and $A \subseteq \mathrm{Diff}(\vec{A})$. q.e.d

Now let's look at the memory requirements to play this game successfully.

Theorem A.5. *Let G be a good news/bad news game with requirements $\vec{Q}$. Then a winning automaton strategy exists for one of the players with memory M such that $|M| \leq \mathrm{lh}(\vec{Q})$.*

That is, the memory required to solve a good news/bad news game is bounded by its difficulty.

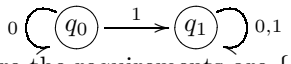
Proof. Each stage can be analysed as a reachability game (a little care must be taken; winning the stage does not necessarily mean winning the game, so not all stage-winning strategies are equally useful). Since reachability games admit memoriless strategies, the memory required will be at most the number of stages (the difficulty of the game). q.e.d

However given an automaton, a good news/bad news game can be constructed with arbitrary difficulty. Is this bound unnecessarily loose? It turns out that it is not, as stated in the following theorem.

Theorem A.6. *For any $n \in \omega$, a good news/bad news game exists for which a winning automaton strategy requires a memory of size at least n. Furthermore, such a game can be shown on an automaton with only two states.*

[2] In fact, if the requirements are non-trivial —in the sense that for no n is $\vec{Q}(n) = Q$— the presentation given is canonical.

Proof. Take the automaton



and the family of games where the requirements are $\{q_0\}$ for the first $2n$ stages then $\{q_1\}$. Player **I** has a winning strategy in each of these games, but any such winning strategy requires a memory of size n. In effect, **I** has to count through the first $2n$ stages. Suppose **II** plays only 0. Then if **I** plays 1 before stage $2n+1$, he has lost. But if he fails to play 1 at *some* point in the game, then likewise he loses. With less than n memory states, at some stage before $2n$ he either plays a 1 and loses, or enters a cycle in which he will continue to play only 0 and thus loses to the all-0 play from **II**. q.e.d

Remark (The connection with parity automata). This type of game can be naturally generalised to the infinite case by taking a requirement of length ω. Once you do so, however, it is no longer necessarily possible to analyse the game using a finitary algorithm (see the following section) or win using a finite-memory strategy (via a generalisation of the argument above).

Suppose instead that we convert a finitary requirement to an infinitary simply by removing from the trace every state that occurs only finitely many times. If a further restriction on requirements is invoked, that the sets appearing are pairwise disjoint, then the result is a parity game (highest parity wins) as described in Definition 2.28.

A.3 Time complexity for solution algorithms

We have an obvious upper bound through the analysis by reachability: using `CG-book` for each stage we get a time complexity of $\mathrm{O}(|E| \cdot |V| \cdot \mathrm{lh}(\vec{Q}))$, where $\langle V, E\rangle$ is the graph of the automaton and $\vec{Q}$ is the requirement. However we can restrict this by noticing that although the requirement can grow without bound, the number of different paths that need to be considered is bounded by the size of the graph.

Theorem A.7. *Let G be a good news/bad news game on an automaton with graph $\langle V, E\rangle$, and let the length of the requirement be n. Then there exists an algorithm `A` solving G such that `A` $\in \mathrm{O}(|E| \cdot |V| \cdot n) \cap \mathrm{O}(|E| \cdot |V|^3)$.*

Proof. There are only $|V|^2$ different point-to-point reachability goals, so an algorithm that computes the reachability for each stage but remembers the results for vertex pairs it has checked before performs the minimum of $|V|^2$ and n reachability analyses. q.e.d

References

[And01] Alessandro Andretta. *Notes on Descriptive Set Theory*. In preparation, June 2001.

[BLRvBar] Stefan Bold, Benedikt Löwe, Thoralf Räsch, and Johan van Benthem, editors. *Infinite Games: Papers of the conference "Foundations of the Formal Sciences V", held in Bonn November 2004*, to appear.

[dB04] Boudewijn de Bruin. *Explaining Games: On the Logic of Game Theoretic Explanations*. PhD thesis, Institute for Logic, Language and Computation, 2004.

[dJLar] Samson de Jager and Benedikt Löwe. Nonmonotone game labellings. In Bold et al. [BLRvBar].

[Far02] Berndt Farwer. ω-automata. In Grädel et al. [GTW02], pages 3–22.

[Fra97] Aviezri S. Fraenkel. Combinatorial game theory foundations applied to digraph kernels. *The Electronic Journal of Combinatorics*, 4(2), 1997.

[Fri81] Harvey Friedman. On the necessary use of abstract set theory. *Advances in Mathematics*, 41(3):209–280, 1981.

[GS53] David Gale and Frank M. Stewart. Infinite games with perfect information. In Harold W. Kuhn and Albert W. Tucker, editors, *Contributions to the Theory of Games II*, volume 28 of *Annals of Mathematical Studies*, pages 245–266. Princeton University Press, 1953.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[HD05] Paul Hunter and Anuj Dawar. Complexity bounds for regular games. To be presented at the 30th International Symposium on Mathematical Foundations of Computer Science, Gdansk, September 2005.

[HMar] Christoph Heinatsch and Michael Möllerfeld. Determinacy in second order arithmetic. In Bold et al. [BLRvBar].

[Hur93] A. J. C. Hurkens. *Borel determinacy without the axiom of choice*. PhD thesis, Kahtolieke Universiteit Nijmegen, 1993.

[Kan03] Akihiro Kanamori. *The Higher Infinite: Large Cardinals in Set Theory from Their Beginnings*. Springer, second edition, 2003.

[Kec95] Alexander Kechris. *Classical Descriptive Set Theory*. Springer-Verlag, 1995.

[Küs02] Ralf Küsters. Memoryless determinacy of parity games. In Grädel et al. [GTW02], pages 95–106.

[LÖ3] Benedikt Löwe. Determinacy for infinite games with more than two players with preferences. Technical Report PP-2003-19, Institute for Logic, Language and Computation, 2003.

[LS05] Benedikt Löwe and Brian Semmes. The extent of constructive game labellings. In Costas Dimitracopoulos, editor, *Proceedings of the 5th Panhellenic Logic Symposium*, pages 93–98, July 2005.

[Mar] Donald A. Martin. Determinacy. Monograph in preparation.

[Mar75] Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975.

[Maz02] René Mazala. Infinite games. In Grädel et al. [GTW02], pages 23–38.