Mathematical Social Sciences 24 (1992) 117–139
North-Holland

Recursively presented games and strategies

Douglas Cenzer

University of Florida, Gainesville, FL 32611-2082, USA

Jeffrey Remmel

University of California–San Diego, La Jolla, CA 92093, USA

Communicated by W.H. Batchelder
Received 30 August 1989
Revised 15 November 1992

The complexity of an infinite game is determined in part by the complexity of the set of winning strategies for the game. In this paper we consider effectively closed, recursively bounded Gale–Stewart games. We give a coding for the winning strategies of such games so that any two different strategies are winning for different sets of games. We show that under our coding of winning strategies, the set of winning strategies can be effectively isomorphic to an arbitrary recursively bounded Π^0_1-class. This strengthens the well-known result that recursively presented games need not have recursive winning strategies. We also show that for every recursively presented game, there is a polynomial time presented game which has the same set of winning strategies. Thus even feasibly presented games need not have recursive solutions. Finally we develop some criteria which ensures that a polynomial time presented game has a polynomial time winning strategy.

Key words: Complexity; infinite game; Gale–Stewart game.

1. Introduction

Recursively bounded Π^0_1-classes arise naturally in the study of recursive combinatorics and recursive algebra. For our purposes, we will define a *recursively bounded* (*r.b.*) Π^0_1 to be the set $[T]$ of all infinite paths through a highly recursive finitely branching tree T. Here a *highly recursive finitely branching tree* is just a recursive subset of $\omega^{<\omega}$, the set of all finite sequences from the natural numbers $\omega = \{0, 1, 2, \dots\}$, which is closed under initial segments and with the property that there is a partial recursive function $f: T \to \omega$ such that, for each node σ of T, $f(\sigma)$ is the number of successors of σ. The basic idea is that given a recursively presented problem, one can often show that the set of solutions of the problem is a recursively

Correspondence to: D. Cenzer, Department of Mathematics, University of Florida, 201 Walker Hall, Gainesville, FL 32611-2082, USA.

bounded Π^0_1-class. For example, the recursively presented problem may be to find a proper coloring of a recursive graph as in Remmel (1986), to find a proper marriage in a recursively presented society as in Manaster and Rosenstein (1972), or to find an ordering of a formally real recursive field as in Metakides and Nerode (1979). Thus the solutions to the three recursively presented problems are, respectively, the set of all proper colorings of the graph, the set of all proper marriages in the society, and the set of all orderings of the formally real field. By showing that the set of solutions to the recursively presented problem is a recursively bounded Π^0_1-class, one shows that the complexity of finding a solution to the recursively presented problem is no more than that of finding an infinite path through a highly recursive tree. One can then try to reverse the process and show that for any highly recursive tree T, there is an instance of the recursively presented problem such that there is an effective one-to-one degree preserving correspondence between the set of infinite paths through T and the set of solutions to the instance of the recursively presented problem. In such a situation one has shown that the problem of finding a solution to the recursively presented problem is effectively equivalent to finding an infinite path through a highly recursive tree. For example, such correspondences have been shown to exist in the case of colorings of recursive graphs by Remmel (1986), in the case of symmetric marriages in a highly recursive society by Manaster and Rosenstein (1972), and in the case of orderings of a formally real recursive field by Metakides and Nerode (1979). Moreover, the possible degrees of elements of recursively bounded Π^0_1-classes have been extensively studied, see for example the work of Jockusch and Soare (1972a,b), so that by exhibiting such correspondences one automatically obtains a wealth of information about the possible degrees of solutions to the recursively presented problem. For a detailed development of the role of Π^0_1-classes in mathematics, see Cenzer and Remmel (1992c).

In this paper we will show that recursively bounded Π^0_1-classes also naturally arise in the study of the winning strategies of effectively presented open or closed infinite games of perfect information. These games were introduced by Gale and Stewart (1953), who showed that all such games are determined. Effective Gale–Stewart games and their winning strategies have been studied in depth by Moschovakis (1980) and others. It is well known that effectively closed games can have arbitrarily complicated hyperarithmetic strategies. In this paper we study a certain subclass of effective Gale–Stewart games which we call recursively bounded, recursively presented games. First we shall show that exactly as in the situation for colorings of graphs, proper marriages of societies, and orderings of formally real fields described above, the problem of finding a winning strategy for a recursively bounded, recursively presented Gale–Stewart game is effectively equivalent to finding a path through a highly recursive tree. That is, we shall show that once we specify a certain natural coding of the winning strategies of such games that eliminates all redundant information, it is easy to see that the set of winning strategies for such a game is a recursively bounded Π^0_1-class. This will be explained in detail below. Conversely, we will show that any given recursively bounded

Π^0_1-class may be represented by the set of winning strategies for some recursively bounded, recursively presented game. Second, we study the complexity of winning strategies in polynomial time presented games. We will show that for any recursively bounded, recursively presented game, there is a recursively bounded, polynomial time presented game with exactly the same set of winning strategies. Thus for recursively bounded Gale-Stewart games, if one restricts one's self to polynomial time presented games, one encounters exactly the same degree of difficulty in finding a winning strategy that one encounters in trying to find a winning strategy for recursively presented Gale-Stewart games.

Next we shall formally define the class of recursively bounded, recursively presented Gale-Stewart games. It is clear that for any highly recursive finitely branching tree T, there will be a recursive function $g(n)$ such that for any finite sequence $(s_0, s_1, \ldots, s_n) \in T$, $s_i \leqslant g(i)$ for all $i \leqslant n$. Each such tree T will determine a recursively bounded, recursively presented Gale-Stewart game $G(T)$, played by two players, Player I and Player II, as follows. The two players alternately play an infinite sequence $(x(0), y(0), x(1), y(1), \ldots)$ of natural numbers with $x(n) \leqslant g(2n)$ and $y(n) \leqslant g(2n+1)$ for each n. Thus Player I begins by playing $x(0) \leqslant g(0)$, then Player II responds with $y(0) \leqslant g(1)$, Player I responds in turn with $x(1) \leqslant g(2)$, and so on ad infinitum. Now Player II is said to win the play z of the game if the resulting infinite sequence z is an infinite path through the tree T, that is, if $z \in [T]$. (In the dual game, we let Player I win if $z \in [T]$.)

Jockusch and Soare (1972a,b) gave a detailed analysis of the possible Turing degrees of elements of r.b. Π^0_1-classes. Once we have proved our main results on the set of winning strategies of r.b. recursively presented Gale-Stewart games, we will be able to apply their results to give an analysis of the complexity of the winning strategies of an infinite recursively bounded, recursively presented Gale-Stewart game. It follows in particular that such games need not have recursive winning strategies.

The definition of a polynomial time tree is not entirely straightforward. We shall discuss our definition of a polynomial time tree in Section 4. Once we have defined polynomial-time trees, a polynomial time presented Gale-Stewart game will just be a game $G(T)$ where T is a polynomial time tree. Then the reduction of a recursively presented game to a polynomial presented game is based on the following result. We will show that for any recursive tree T, there is a polynomial-time tree T' such that $[T] = [T']$. Then clearly the games $G(T]$ and $G(T']$ must have the same winning strategies. It will then follow that even recursively bounded, polynomial time presented games need not have any recursive winning strategies. Finally we will give various conditions on polynomial time trees which will imply that the tree will have an infinite path of a given complexity (either polynomial time, exponential time, double exponential time, or non-deterministic polynomial time). This will give rise to conditions on the polynomial time tree T which will imply that $G(T)$ is guaranteed to have a winning strategy of a given complexity. The complexity of the game and of the strategy depends on the representation of the natural numbers. We

will consider both the binary and the unary, or tally, representation.

The paper is organized as follows. Section 2 contains preliminaries and notation. Section 3 contains the coding of the winning strategies and the results that the set of winning strategies is always a recursively bounded Π^0_1-class and may in fact represent an arbitrary recursively bounded Π^0_1-class. Section 4 contains the results on recursive versus polynomial time trees and games.

2. Preliminaries

Some definitions are needed. Let Σ be a finite alphabet. Then Σ^* denotes the set of finite strings of letters from Σ and Σ^ω denotes the set of infinite sequences, where $\omega = \{0, 1, 2, \dots\}$. In particular, each natural number n may be represented in unary form by the string $\mathrm{tal}(n) = 1^n$ and in (reverse) binary form by the string $\mathrm{bin}(n) = i_0 \dots i_k$, where $n = i_0 + i_1 \cdot 2 + \cdots + i_k \cdot 2^k$.

We let $\mathrm{Tal}(\omega) = \{\mathrm{tal}(n) : n \in \omega\}$ and $\mathrm{Bin}(\omega) = \{\mathrm{bin}(n) : n \in \omega\}$. Both sets are included in $\{0,1\}^*$.

For a string $\sigma = (\sigma(0), \sigma(1), \dots, \sigma(n-1))$, $|\sigma|$ denotes the length n of σ. The empty string has length 0 and will be denoted by $\emptyset$. A constant string σ of length n will be denoted k^n. For $m < |\sigma|$, $\sigma \restriction m$ is the string $(\sigma(0), \dots, \sigma(m-1))$; σ is an *initial segment* of τ (written $\sigma < \tau$) if $\sigma = \tau \restriction m$ for some m. The *concatenation* $\sigma * \tau$ (or sometimes just $\sigma\tau$) is defined by

$$\sigma * \tau = (\sigma(0), \sigma(1), \dots, \sigma(m-1), \tau(0), \tau(1), \dots, \tau(n-1)),$$

where $|\sigma| = m$ and $|\tau| = n$; in particular we write $\sigma^\frown a$ for $\sigma * (a)$.

A *tree* T over Σ^* is a set of finite strings from Σ^* which is closed under initial segments. We say that $\tau \in T$ is an immediate successor of a string $\sigma \in T$ if $\tau = \sigma^\frown a$ for some $a \in \Sigma$. A recursive tree T is said to be *highly recursive* if there is a partial recursive function $f : T \to \omega$ such that, for any $\sigma \in T$, σ has at most $f(\sigma)$ immediate successors in T. For any tree T, an *infinite path* through T is a sequence $(x(0), x(1), \dots)$ such that $x \restriction n \in T$ for all n. Let $[T]$ be the set of infinite paths through T. A subset X of Σ^ω is a Π^0_1-class if $X = [T]$ for some recursive tree T; if T is highly recursive, then $X = [T]$ is said to be a *recursively bounded* Π^0_1-class.

Our basic computation model is the standard multitape Turing machine of Hopcroft and Ullman (1969). Note that there are different heads on each tape and that the heads are allowed to move independently. This implies that a string σ can be copied in linear time.

Let $t(n)$ be a function on natural numbers such that $t(n) \geqslant n$. A (deterministic) Turing machine M is said to be *$t(n)$-time bounded* if each computation of M on inputs of size n where $n \geqslant 2$ is completed in at most $t(n)$ steps. If t can be chosen to be a polynomial then M is said to be *polynomial-time bounded*. A function $f(x)$ on strings is said to be in $DTIME(t)$ if there is a $t(n)$-time bounded Turing machine M which computes $f(x)$. A set of strings or a relation on strings is in $DTIME(t)$ if its

characteristic function is in $DTIME(t)$. We let

$P = \bigcup_i \{DTIME(n^i) : i \geqslant 0\}$,

$DEXT = \bigcup_{c \geqslant 0} \{DTIME(2^{c \cdot n}\}$,

$DEXPTIME = \bigcup_{c \geqslant 0} \{DTIME(2^{n^c}\}$, and

$DDOUBEXT = \bigcup_{c \geqslant 0} \{DTIME(2^{2^{c \cdot n}}\}$.

We say that a function $f(x)$ is *polynomial time* if $f(x) \in P$, is *exponential time* if $f(x) \in DEXT$, and is *double exponential time* if $f(x) \in DDOUBEXT$.

We refer the reader to Rogers (1967) for the basic definitions of recursion theory. Let ϕ_i be the partial recursive function computed by the ith Turing machine M_i. Given a string $\sigma \in \{0,1\}^*$, we write $\phi_i^s(\sigma)\downarrow$ if M_i gives an output in s or fewer steps when started on input string σ. Thus the function ϕ_i^s is uniformly polynomial time. We write $\phi_e(\sigma)\downarrow$ if $(\exists s)(\phi_e^s(\sigma)\downarrow)$ and $\phi_e(\sigma)\uparrow$ if not $\phi_e(\sigma)\downarrow$. Given two sets A and B, we write $A <_T B$ if A is Turing reducible to B and we write $A \equiv_T B$ if both $A <_T B$ and $B <_T A$. We let $\mathbf{0}$ denote the degree of the recursive sets and we let $\mathbf{0}'$ denote the degree of the jump of the recursive sets.

Let Γ be some complexity class of sets (and functions), such as partial recursive, primitive recursive, exponential time, polynomial time (or p-time). We say that a set or function is Γ-computable if it is in Γ.

3. Recursively bounded Π_1^0-classes and infinite games

In this section we show that the set of winning strategies for a recursively presented, recursively bounded game is always a recursively bounded Π_1^0-class and can represent an arbitrary recursively bounded Π_1^0-class. If the players are restricted to playing either 0 or 1 at every turn, we say that the game is a $\{0,1\}$-game. Now it is easy to see that any recursively presented, recursively bounded game can always be coded by a $\{0,1\}$-game by using several turns in the $\{0,1\}$-game to code one turn in the ω game. For simplicity, we shall only consider $\{0,1\}$-games in this section.

In general any subset X of $\{0,1\}^\omega$ determines a $\{0,1\}$-game $G(X)$ where Player II wins if the play of the game is in the set X. A *strategy* for Player II is a (partial) function θ from $\{0,1\}^{<\omega}$ into $\{0,1\}$. For any play $x = (x(0), x(1), \ldots)$ of the game by Player I, the play $\theta(x)$ of the game when θ is applied to x is given by $(x(0), y(0), x(1), \ldots)$, where, for each n, $y(n) = \theta(x(0), y(0), \ldots, y(n-1), x(n))$. The strategy θ is said to be a *winning strategy for Player II in the game* $G(X)$ if, for any play x of the game by Player I, $\theta(x) \in X$. The notion of a strategy and a winning strategy for Player I is similarly defined. The game $G(X)$ is said to be *determined* if one of the two players has a winning strategy. The result that all finite games of perfect information are determined is due to Zermelo; see von Neumann and Morgenstern (1947) for details. Now if X is a clopen subset of $\{0,1\}^\alpha$, then the game $G(X)$ is essentially finite, since a clopen set is a finite union of intervals. Thus all clopen games $G(X)$ are determined. Gale and Stewart (1953) showed that all

games $G(X)$, where X is either open or closed, are determined.

A winning strategy for a clopen game is basically a finite object and is therefore recursive. It follows that every clopen game has a recursive strategy and that the set of winning strategies for a clopen game can be viewed as a clopen set. Now a recursive subset X of $\{0,1\}^\omega$ is simply a clopen set. Thus recursive games are not very complicated.

The effective version of the Gale-Stewart theorem is that every effectively closed game is determined. Now an effectively closed subset X of $\{0,1\}^*$ is simply a Π^0_1-class, that is, $X=[T]$, the set of infinite paths through some recursive tree $T\subseteq 2^{<\omega}$. In general, for any recursive tree $T\subseteq\omega^{<\omega}$, let $G(T)$ be the game $G([T])$.

We will now describe the coding of a strategy θ for a game $G(T)$. A strategy ν for the game $G(T)$ is usually viewed as a total function from $\{0,1\}^\omega$ into $\{0,1\}$; see, for example, Moschovakis (1980). This has obvious drawbacks for our situation. First observe that, for any strategy θ of Player II, if $\theta(0))=0$, then $\theta((010))$ and $\theta((011))$ are irrelevant, because no play of the game in which Player II follows the strategy θ can ever pass through (01). This means that for any winning strategy θ of Player II, there are continuum many (total) strategies which are equivalent to θ but differ on irrelevant inputs. In fact, there are continuum many (total) strategies θ' which give the exact same result $\theta'(x)=\theta(x)$ for all plays x of Player I. This means, of course, that the set of (total) winning strategies for any game must be uncountable so that we could not, using total strategies, represent every recursively bounded Π^0_1-class as the set of winning strategies for an effectively closed game. For our purposes, we will code up the strategy so that the extra information not needed for the play of the game is eliminated.

Let $\tau_0,\tau_1,\ldots$ enumerate the non-empty elements of $\{0,1\}^{<\omega}$ in order, first by length and then lexicographically, so that $\tau_0=(0)$, $\tau_1=(1)$, $\tau_2=(00),\ldots$. In general, $|\tau_k|=n$ if and only if $2^n-2\leqslant k<2^{n+1}-2$. For each $\tau\in\{0,1\}^*$, let $n(\tau)$ be the unique n such that $\tau=\tau_n$. Then an arbitrary sequence $z=(z(0),z(1),\ldots)\in\{0,1\}^\omega$ codes a strategy θ_z for Player II in the following manner. For any play $x=(x(0),x(1),\ldots)$ of Player I, the strategy θ_z produces the following response $y=(y(0),y(1),\ldots)$ by Player II. First, $y(0)=z(n((x(0)))$ and for any k, $y(k+1)=z(n)$, where $\tau_n=(x(0),\ldots,x(k))$, that is, $\theta_z((x(0),y(0),\ldots,y(k-1),x(k))=z(n)$. Thus $z(0)=\theta_z((0))$, $z(1)=\theta_z((1))$, $z(2)=\theta_z(0,\theta_z(0),0)$ and so on. It is clear that the result $\theta_z(x)$ of applying this strategy to a play $x=(x(0),x(1),\ldots)$ of the game by Player I can be computed from x and z by a recursive function. For a finite sequence $z\upharpoonright n=(z(0),\ldots,z(n-1))$, $\theta_{z\upharpoonright n}$ is a *partial strategy* which, applied to any partial play $x\upharpoonright m+1=(x(0),\ldots,x(m))$ of Player I with $n(x\upharpoonright m)<n$, gives a partial response $\theta_{z\upharpoonright n}(x(0),y(0),\ldots,y(m-1),x(m))=y(m)$, where for all $r\leqslant m$, $y(r)=z(k_r)$ if $n((x(0),\ldots,x(r))=k_r$. (Note our ordering of $\{0,1\}^*$ ensures that $k_r<n$ for all $r\leqslant m$.) Now, for any tree $T\subseteq 2^{<\omega}$, let $WS(T)$ be the set of codes $z=(z(0),z(1),\ldots)\in\{0,1\}^\omega$ for winning strategies of Player II in the game $G(T)$.

We first make precise our claim that this coding removes all of the unnecessary information from a strategy.

Proposition 3.1. *For any $z_0, z_1 \in \{0,1\}^\omega$, if $z_0 \neq z_1$, then there is a clopen game G such that θ_{z_0} is a winning strategy for Player II for the game G but θ_{z_1} is not a winning strategy for the game G for Player II.*

Proof. Let n be the least such that $z_0(n) \neq z_1(n)$; we can assume without loss of generality that $z_i(n) = i$ for $i = 0, 1$. Let $\tau_n = (x(0), \ldots, x(r))$ and let $y(m) = \theta_{z_0}((x(0), y(0), \ldots, y(m-1), x(m))$, for $m < r$, recursively define the partial play of the game where Player I plays the sequence $(x(0), \ldots, x(m))$ and Player II follows the strategy θ_{z_0}. It follows from the minimality of n that the strategy θ_{z_1} will produce the same partial play $(x(0), y(0), \ldots, x(r-1), y(r-1))$. However, at the rth move, strategy θ_{z_0} tells Player II to play a 0 in response to $x(r)$, whereas strategy θ_{z_1} tells Player II to play a 1. Now define the clopen set X to consist of all elements of $\{0,1\}^\omega$ which either do not begin with $(x(0), y(0), \ldots, x(r-1), y(r-1), x(r))$ or which begin with $(x(0), y(0), \ldots, x(r-1), y(r-1), x(r), 0)$. It is clear that θ_{z_0} is a winning strategy for $G(X)$ for Player II but θ_{z_1} is not. □

Theorem 3.2. *For any recursively bounded, recursively presented game $G(T)$, the set $WS(T)$ of winning strategies of Player II for the game $G(T)$ is a Π^0_1-class.*

Proof. We will assume that $G(T)$ is a $\{0,1\}$-game. (The proof for an arbitrary recursively bounded game is an easy generalization.) For any recursive tree T, we will define a recursive tree Q such that $WS(T) = [Q]$, as follows. First $\emptyset$ is in Q and then for any $\sigma = (z(0), \ldots, z(n-1))$, $\sigma \in Q$ if and only if, for all sequences $\nu = (x(0), \ldots, x(r-1))$ where $n(\nu) < n$, the result of applying the partial strategy θ_σ coded by σ to the partial play ν is in T. □

We should note that having established the fact that the set of winning strategies for a recursively bounded, recursively presented game corresponds to a r.b. Π^0_1-class, we can derive a number of immediate corollaries, due to the work of Jockusch and Soare in (1972a,b). For example, the following is true.

Corollary 3.3. *For any recursively bounded, recursively presented closed game of perfect information for which Player II has a winning strategy:*

(i) *There is a winning strategy θ for Player II whose Turing jump is recursive in $\mathbf{0}'$, the Turing jump of the empty set.*

(ii) *If Player II has only finitely many winning strategies, then each winning strategy is recursive.*

(iii) *There is a winning strategy for Player II which is in some r.e. degree.*

(iv) *There are winning strategies θ_1 and θ_2 such that any function recursive in both θ_1 and θ_2 is recursive.*

(v) *If Player II has no recursive winning strategy, then for any countable sequence of non-zero degrees $\{a_i\}$, Player II has continuum many winning strategies*

which are mutually Turing incomparable and at the same time Turing incomparable with each a_i.

(vi) *If Player II has no recursive winning strategy, then there is a non-zero r.e. degree a such that Player II has no winning strategy recursive in a.* □

Finally, we should note that the correspondence between winning strategies for a closed game $G(T)$ and infinite paths in the closed set $WS(T) = [Q]$ constructed as outlined above can also be viewed as a means of extending the proof that all finite games are determined to a proof that all Gale-Stewart games are determined. Observe first that any open set U is a union of intervals $I(\sigma) = \{x : \sigma < x\}$. Now any infinite closed game $G(X)$ can be truncated after n moves to give a finite game where Player I wins the play σ if $I(\sigma) \cap X = \emptyset$. Thus if we assume that Player I does not have a winning strategy for the game $G(X)$, then Player I does not have winning strategies for any of the truncated finite games. Then by the finite game theorem of Zermelo, Player II must have winning strategies for every truncated game. These finite strategies will correspond to finite paths through the tree Q. It then follows that Q is an infinite tree contained in $\{0,1\}^{<\omega}$ and hence by Konig's Lemma, there is an infinite path through Q, that is, there is a winning strategy for Player II for the game $G(X)$. This shows that one of the two players must have a winning strategy so that the closed game $G(X)$ is determined.

Now we will consider the reverse direction of this correspondence. That is, suppose we are given a highly recursive tree. Is it possible to construct a recursively bounded, recursively presented game $G(X)$ so that the set of winning strategies for Player II for the game $G(X)$ correspond to the infinite paths through the tree. Formally, when we say that there is an *effective one-to-one degree-preserving correspondence* between two Π^0_1-classes $[Q]$ and $[T]$, we mean that there is a recursive function Φ such that the map $y = \Phi(x)$ is a one-to-one degree preserving map from $[Q]$ to $[T]$. Here a recursive functional acting on input x can be viewed as an oracle Turing machine M with oracle x such that, for any n, $y(n) = M^x(n)$.

Theorem 3.4. *For any highly recursive finitely branching tree Q, there is an effectively closed subset X of $\{0,1\}^\omega$ and an effective one-to-one degree preserving correspondence between the recursively bounded Π^0_1-class $[Q]$ of infinite paths through Q and the class of winning strategies for the effectively closed game $G(X)$.*

Proof. We may assume without loss of generality that the tree Q is a subset of $\{0,1\}^*$ (see Remmel, 1986, p. 187). The set X will be defined as the set of infinite paths through the tree T, which will be defined recursively. Our basic idea is that each path $\Pi = (\pi(0), \pi(1), \ldots)$ should correspond to a strategy θ_Π which acts as follows. Given any partial play, $((x(0), \ldots, x(m))$ of Player I, θ_Π will respond with

$$\theta_\Pi((x(0), y(0)), \ldots, y(m-1), x(m))) = y(m),$$

where $y(m) = 0$ if $x(i) = 0$ for any $i \leqslant m$ and $y(m) = \pi(m)$ if $x(i) = 1$ for all $i \leqslant m$. Thus

whenever Player I plays a 0, then ever after θ_Π will respond with a 0 and if Player I plays all 1's, then θ_Π will respond by reproducing the path Π. It is easy to see that when we code the strategy θ_Π via a sequence $z=(z(0), z(1), \dots)$ that z will have the same Turing degree as Π. Thus the correspondence $\Pi \to \theta_\Pi$ will be an effective one-to-one degree preserving correspondence. Thus all we need to do is recursively define a highly recursive tree $T \subseteq 2^{<\omega}$ so that $WS(T)=\{z \in \{0,1\}^\omega : z$ is a code of a θ_Π for $\Pi \in Q\}$. Begin by putting $(0,0)$ in T, leaving $(0,1)$ out, and, for $i=0,1$, put $(1,i) \in T \Leftrightarrow (i) \in Q$. (Note that putting $(0,0)$ in T but leaving $(0,1)$ out of T corresponds to ensuring that if Player I starts with a 0, then the only winning strategies we allow must respond with a 0. Putting $(1,i)$ in T for $i \in Q$ says that a winning strategy must respond to Player I's first move of 1 with a node at level one in Q. Similar remarks will apply to the subsequent nodes we put in T.) Then, for each n and each $\tau=(x(0), y(0), \dots, x(n), y(n)) \in T$, do the following:

(1) If $x(k)=0$ for some $k \leqslant n$, then put $\tau^\frown i^\frown 0 \in T$ and leav $\tau^\frown i^\frown 1$ out of T for $i=0,1$.

(2) If $x(k)=1$ for all $k \leqslant n$, then put $\tau^\frown 0^\frown 0 \in T$, leave $\tau^\frown 0^\frown 1$ out of T, and put $\tau^\frown 1^\frown i \in T$ if and only if $(y(0), \dots, y(n), i) \in Q$.

It easily follows from the definition of T that for any $\Pi=(\pi(0), \pi(1), \dots) \in [Q]$, θ_Π is a winning strategy for Player II for the game $G(T)$. Now suppose that θ is a winning strategy for Player II for $G(T)$. Then we can define a $\Pi=(\pi(0), \pi(1), \dots) \in [Q]$ such that $\theta=\theta_\Pi$ by recursion as follows. For each n, let $\pi(n)=\theta((1, \pi(0), 1, \pi(1), \dots, 1, \pi(n-1), 1))$. It is easy to see from our definition of T that for all $n \geqslant 0$, $(\pi(0), \pi(1), \dots, \pi(n))$ is in Q so that Π will be an infinite path through Q. Then Π codes enough information to compute the values of θ on all inputs of the form $(1, \pi(0), \dots, 1, \pi(n-1), 1)$ and the other values of θ are determined by the definition of T to be 0. This shows that $\theta=\theta_\Pi$. Thus the correspondence $\Pi \to \theta_\Pi$ is our desired effective one-to-one degree preserving correspondence between $[Q]$ and $WS(T)$. □

Once we have Theorem 3.4, we again can apply known results about the degrees of elements of Π^0_1-classes to derive results about the degrees of winning strategies of r.b. recursively presented Gale-Stewart games. All the results in Corollary 3.5 below follow from the work of Jockusch and Soare (1972a,b) except part (vi) which follows from the work of Kucera (1986).

Corollary 3.5.

(i) *There exists a recursively bounded, recursively presented Gale-Stewart game G_1 such that G_1 has no recursive winning strategy.*

(ii) *There exists a recursively bounded, recursively presented Gale-Stewart game G_2 such that any two distinct recursive winning strategies for G_2 are Turing incomparable.*

(iii) *If $\boldsymbol{a}$ is a Turing degree and $\mathbf{0}<_T \boldsymbol{a}<_T \mathbf{0}'$, then there exists a recursively bounded, recursively presented Gale-Stewart game G_3 such that G_3 has a winning*

strategy of degree $\boldsymbol{a}$ *but* G_3 *has no recursive winning strategy.*

(iv) *There exists a recursively bounded, recursively presented Gale–Stewart game* G_4 *such that if* $\boldsymbol{a}$ *is the degree of any winning strategy of* G_4 *and* $\boldsymbol{b}$ *is a r.e. degree with* $\boldsymbol{a} <_T \boldsymbol{b}$, *then* $\boldsymbol{b} \equiv_T \boldsymbol{0}'$.

(v) *If* $\boldsymbol{c}$ *is any r.e. degree, then there exists a recursively bounded, recursively presented Gale–Stewart game* G_5 *such that the set of r.e. degrees which contain winning strategies of* G_5 *equals the set of r.e. degrees* $\geqslant_T \boldsymbol{c}$.

(vi) *There exists a recursively bounded, recursively presented Gale–Stewart game* G_6 *such that if* W *is a winning strategy for* G_6, *where* $W <_T \boldsymbol{0}'$, *then there exists a non-recursive r.e. set* A *such* $A <_T W$. □

Remark. The set of winning strategies for Player I, the open player in the game $G(T)$, can also be coded and seen to be a recursively bounded Σ^0_1-class, that is, the complement of a r.b. Π^0_1-class. It can also be shown that an arbitrary r.b. Π^0_1-class can be represented by the set of non-winning strategies of Player I in some recursively presented game $G(T)$. These results are less interesting, since the corresponding problem of finding a winning strategy for the open player always has a recursive solution. (In fact, there is always an interval $I(\sigma)$ of solutions and therefore a solution which is almost constant and thus linear time.)

4. Feasible trees, games and strategies

In this section we introduce the concept of a polynomial time tree and a p-time presented game. Since recursively presented games did not necessarily have recursive strategies, one basic question to be answered is whether a p-time presented game is more likely to have a recursive strategy. We will consider conditions under which a p-time strategy can be found, as well as weaker conditions which imply the existence of exponential or of double-exponential-time strategies.

For a detailed development of the notion of p-time versus recursive models, the reader is referred to Cenzer and Remmel (1991, 1992a), where in particular p-time orderings, Boolean algebras and Abelian groups are studied. Feasibly presented combinatorial problems are studied in Cenzer and Remmel (1992c) and the graph coloring problem in particular in Cenzer and Remmel (1992b).

Since the winning strategies of an effectively closed game correspond to infinite paths through some recursive tree, we are interested in the possible complexity of recursive trees and of the paths through those trees. First we shall consider the definitions of a p-time tree T and of a p-time path through T. We think of a recursive tree T as a set of finite sequence $(n_0, \ldots, n_{k-1})$ of natural numbers and of an infinite path $(n_0, n_1, \ldots)$ through T as a function from the natural numbers into the natural numbers which maps i to n_i. We shall define two natural representations of T which will be useful for the study of the complexity of trees and paths through trees. First we define the *binary representation* of T, $\mathrm{bin}(T)$, as the set of finite

strings $\{(\text{bin}(n_0), \ldots, \text{bin}(n_{k-1})) : (n_0, \ldots, n_{k-1}) \in T\}$. We also define the *tally representation* of T, $\text{tal}(T)$, to be the set of strings $\{(\text{tal}(n_0), \ldots, \text{tal}(n_{k-1}) : (n_0, \ldots, n_{k-1}) \in T\}$. The strings in $\text{bin}(T)$ and $\text{tal}(T)$ are over the finite alphabet $\{0, 1, ',', '(', ')'\}$ which has symbols for the comma and the left and right parentheses. We say that T is *p-time in binary* if $\text{bin}(T)$ is a p-time subset of Σ^*. Similarly we say T is *p-time in tally* if $\text{tal}(T)$ is a p-time subset of Σ^*. Since $\text{bin}(n)$ can be computed in p-time from $\text{tal}(n)$, it follows that if $\text{bin}(T)$ is p-time, then $\text{tal}(T)$ is also p-time. Given an infinite path $x = (n_0, n_1, \ldots)$ through T, the binary representation of x is the function $\text{bin}(x)$ from $\text{Tal}(\omega)$ to $\text{Bin}(\omega)$ defined by $\text{bin}(x)(\text{tal}(i)) = \text{bin}(n_i)$. The tally representation of x, $\text{tal}(x)$, is similarly defined by $\text{tal}(x)(\text{tal}(i)) = \text{tal}(n_i)$. Then we say that x is a *p-time path in binary* if the function $\text{bin}(x)$ is the restriction of a p-time function from $\{0,1\}^*$ to $\{0,1\}^*$; we say that x is *p-time in tally* if $\text{tal}(x)$ is the restriction of a p-time function from $\{0,1\}^*$ to $\{0,1\}^*$. It is clear that if x is p-time in tally, then x is also p-time in binary, since $\text{bin}(x)(\text{tal}(i))$ can be computed from $\text{tal}(x)(\text{tal}(i))$ for each i. The reason for using $\text{Tal}(\omega)$ for the domain of $\text{bin}(x)$ is the following. For any path x, x is recursive if and only if the initial segment function $\bar{x}$ is recursive, where $\bar{x}(i) = (n_0, \ldots, n_{i-1})$. We want to have a similar result for p-time paths, and this would be impossible if $\bar{x}$ had to map $\text{bin}(i)$, which has length roughly $\log_2(i)$, to a string which must have length at least i. Similar definitions can be given for other notions of complexity, such as exponential time, non-deterministic polynomial time (NP), etc.

Recall that a tree $T \subset \omega^{<\omega}$ is highly recursive if there is a recursive function f such that, for any node $\sigma \in T$, $f(\sigma)$ is the number of immediate successors $\sigma^\frown i$ of σ in T. Now given the number of successors of a node, we can search through all the possible immediate successors and find the largest one. Thus we can find a recursive function g such that $g(\sigma)$ is the largest i such that if $\sigma = (\sigma_0, \ldots, \sigma_n) \in T$, then $(\sigma_0, \ldots, \sigma_n, i)$ is in T. Finally, we can also compute recursively the sequence $h(\sigma) = (i_1, \ldots, i_d)$ which lists all i such that $(\sigma_0, \ldots, \sigma_n, i)$ is in T in increasing order. It is clear that f is recursive if and only if g is recursive and if and only if h is recursive. The situation is different for p-time complexity. Consider first the binary representation of T so that we identify a node $\sigma \in T$ with a sequence of numbers in $\text{Bin}(\omega)$. It is not hard to see that if h is p-time, then both f and g are p-time. However, these are the only relations which are guaranteed to hold between the three functions. To see this, consider the following three examples.

(1) Define the sequence $x_0, x_1, \ldots$ of natural numbers by letting $x_0 = 1$ and, for each n, $x_{n+1} = 2^{x_n}$ and let $T = \{(x_0, \ldots, x_{i-1}) : i \in \omega\}$. Then the tree T is p-time and f is p-time, since $f(\sigma) = 1$ for all $\sigma \in T$. However, the function g cannot be p-time since, if $\sigma = (x_0, \ldots, x_n)$, then in the binary representation $|\sigma| \leqslant 3\,|x_n|$, whereas $|x_{n+1}| = 2^{|x_n|}$.

(2) Define the tree T_1 recursively by putting $\emptyset \in T_1$ and, for any $\sigma = (x_0, \ldots, x_{n-1}) \in T_1$, putting $\sigma^\frown i \in T_1$ if and only if $i \leqslant 1 + x_0 + \cdots + x_{n-1}$. T_1 is clearly p-time and the function g is also p-time since $g((x_0, \ldots, x_{n-1})) = 1 + x_0 + \cdots + x_{n-1}$. However, the function h which lists the immediate successors of any node is not

p-time because, for any n, if $\sigma=(1,2,4,\ldots,2^n)$, then $h(\sigma)=(0,1,\ldots,2^{n+1})$, so that in the binary representation $|h(\sigma)|>2^{n+1}$, whereas $|\sigma|=(n+2)(n+3)/2$.

(3) For this example, we will appeal to the intractability of the well-known P = NP conjecture. That is, we will define a p-time tree T_2 for which the function g is p-time and such that if the associated function f were p-time, then the P = NP conjecture would be true. The tree T_2 will be defined so that $\sigma=(n_0,n_1,\ldots,n_{2k+1})\in T_2$ if and only if, for each $i\leqslant k$, $\mathrm{bin}(n_{2i})$ codes a graph on i vertices and $\mathrm{bin}(n_{2i+1})$ either codes a Hamiltonian path on the graph coded by $\mathrm{bin}(n_{2i})$ or is a string of 1's of the appropriate length. Now a graph G_i on i vertices $v_1,\ldots,v_i$ is determined by a set of unordered pairs (v_r,v_s) of vertices (the edges of the graph). There are $\binom{i}{2}=i(i-1)/2$ possible edges in G_i and these may be lexicographically ordered so that a sequence $e_1,e_2,\ldots,e_{\binom{i}{2}}$ codes the graph G_i where, for all $t\leqslant\binom{i}{2}$, $e_t=1$ if G_i has the tth edge and $e_t=0$ otherwise. Of course the (reverse) binary representation $\mathrm{bin}(n_i)$ must end with a 1, so the graph G_i will actually be coded by the string $(e_0,\ldots,e_{\binom{i}{2}},1)$. Observe that this code for G_i will always be a string of length $1+\binom{i}{2}$ and that any binary number $\mathrm{bin}(n)$ of length $1+\binom{i}{2}$ will code a graph on i vertices. Now a Hamiltonian path on G_i is a permutation $(v_{r_0},v_{r_1},\ldots,v_{r_i})$ of the vertices such that there is an edge joining v_{r_t} with $v_{r_{t+1}}$ for all $t<i$. Such a path will be coded by the binary sequence $0^{r_0}10^{r_1}1\ldots0^{r_i}1$, which will always be a binary number of length $\binom{i+1}{2}$ and a binary number $\mathrm{bin}(n)$ of length $\binom{i+1}{2}$ will code a possible Hamiltonian path on a graph of i vertices if and only if $\mathrm{bin}(n)$ has exactly i 1's. It is easy to see that there is a p-time algorithm which will decide, given two binary numbers $\mathrm{bin}(n)$ and $\mathrm{bin}(m)$, whether $\mathrm{bin}(n)$ has length $\binom{i}{2}+1$ for some $i<|\mathrm{bin}(n)|$ and therefore codes a graph G on i vertices and whether $\mathrm{bin}(m)$ code a Hamiltonian path on that graph. The tree T_2 can now be defined by putting $\sigma=(\mathrm{bin}(n_0),\ldots,\mathrm{bin}(n_{2k+1})\in T_2$ if and only if, for each $i<k$, $\mathrm{bin}(n_{2i})$ codes a graph G_i on i vertices and $\mathrm{bin}(n_{2i+1})$ either codes a Hamiltonian path on G_i or equals $\mathrm{tal}(\binom{i+1}{2})$. It follows from the discussion above that T_2 is a p-time tree. Now the function g for this tree is p-time since, for any $\sigma=(\sigma_0,\ldots,\sigma_t)\in T_2$, we have $g(\sigma)=2^{\binom{i+1}{2}}-1$ if $t=2i$ and $g(\sigma)=2^{1+\binom{i}{2}}-1$ if $t=2i-1$. (In each case, $g(\sigma)$ is just a string of 1's of the right length.) On the other hand, the function f associated with the tree T_2 has the property that for any $\sigma=(\sigma_0,\ldots,\sigma_{2i})\in T_2$, $f(\sigma)=1$ if and only if the graph G_i coded by σ_{2i} has no Hamiltonian path. Now suppose that f were p-time and let $\mathrm{bin}(n)$ be a code for a finite graph on k vertices. For all $i<k$, let $n_{2i}=2^{\binom{i+1}{2}}-1$ and let $n_{2i+1}=2^{1+\binom{i+1}{2}}-1$. Finally, let $\sigma=(\mathrm{bin}(n_0),\ldots,\mathrm{bin}(n_{2k-1}),\mathrm{bin}(n))$. Then the sequence σ can be computed from $\mathrm{bin}(n_0)$ in p-time and G has a Hamiltonian path if and only if $f(\sigma)>1$. It follows that if f were p-time, then the Hamiltonian path problem would be p-time. But it is well known that the Hamiltonian path problem is NP-complete. (See Garey and Johnson, 1978, for an explanation of NP-completeness and the P = NP problem.) Thus we have demonstrated that if the function f associated with the tree T_2 were p-time, then P = NP would be true.

Now the situation is slightly different for tally representation of T where we identify a $\sigma\in T$ with a sequence of numbers in $\mathrm{Tal}(\omega)$. Once again it is easy to see that

if h is p-time, then f and g are p-time. Moreover, example (1) above will still show that f may be p-time without g being p-time. However in this case, if T is p-time in tally and g is p-time, then h is also p-time. To see this, suppose $\sigma=(\sigma_0,\ldots,\sigma_n)$. Note that to find $h(\sigma)$, we need only check whether $(\sigma_0,\ldots,\sigma_n,i)\in T$ for $i\leqslant g(\sigma)$. Now in the tally representation, $|(\sigma_0,\ldots,\sigma_n,0)|<|(\sigma_0,\ldots,\sigma_n,1)|<\cdots<|(\sigma_0,\ldots,\sigma_n,g(\sigma))|$. Then if it takes $q(|\sigma|)$ steps to check whether $\sigma\in\mathrm{tal}(T)$ for each $\sigma=(\mathrm{tal}(\sigma_0),\ldots,\mathrm{tal}(\sigma_n))$, then it will take approximately

$$\sum_{i=0}^{|(\sigma_0,\ldots,\sigma_n,g(\sigma))|} q(i)\leqslant q(|(\sigma_0,\ldots,\sigma_n,g(\sigma))|)^2$$

steps to check whether $(\sigma_0,\ldots,\sigma_n,i)\in T$ for $i\leqslant g(\sigma)$. Thus it is easy to see that we can find $h(\sigma)$ in p-time in the tally representation of σ.

We say that a tree T is *locally p-time* in binary (resp. in tally) if all three of the functions defined above are p-time in binary (resp. tally). In the case that T is not itself p-time, then we will say that T is locally p-time if each of the functions is the restriction to T of a function which is p-time (in binary or tally).

Next we will show that if T is locally p-time, then T is also p-time. The same argument works for both binary and tally. Let Q be either $\mathrm{bin}(T)$ or $\mathrm{tal}(T)$ and suppose that the function h associated with Q is p-time. Given a sequence $\sigma=(\sigma_0,\ldots,\sigma_k)$, here is the procedure for testing whether $\sigma\in\mathrm{tal}(T)$. Begin by computing $h(\emptyset)=(\tau_1,\ldots,\tau_d)$ and checking whether $\sigma_0=\tau_i$ for some $i\leqslant d$. Then, for $j<k$ in turn, compute $h(\sigma_0,\ldots,\sigma_j)$ and check to see that σ_{j+1} is in this list. Suppose that $h(\tau)$ may be calculated in time $p(|\tau|)$, where p is some polynomial. Then since each $(\sigma_0,\ldots,\sigma_j)$ is a substring of σ, we see that we can do each of the h computations in time no greater than $p(|\sigma|)$. To read the resulting list of possible successors of $(\sigma_0,\ldots,\sigma_j)$ and compare each one with σ_{j+1} can then be done in time at most $(c-1)p(|\sigma|)$ for some fixed constant c. Thus each step of the procedure takes time at most $cp(|\sigma|)$. Now there are k such steps and $k\leqslant|\sigma|$, so that the entire procedure takes time at most $c|\sigma|p(|\sigma|)$, which is again a polynomial function of $|\sigma|$.

The functions f, g and h describe the behavior of the tree at a particular node. Now sometimes we need to have a global bound as well. Note that for a recursively bounded tree, there is a recursive function p such that, for all natural numbers k and all $\sigma=(n_0,\ldots,n_k)\in T$, $n_k\leqslant p(k)$. We will say that a tree T is *p-time bounded in binary* if there is a p-time function p such that, for all natural numbers k and all $\sigma=(n_0,\ldots,n_k)\in T$, $|\mathrm{bin}(n_k)|\leqslant p(1^k)$. A tree T is *p-time bounded in tally* if there is a p-time function p such that for any $\sigma=(n_0,\ldots,n_k)\in T$, we always have $n_k=|\mathrm{tal}(n_k)\leqslant p(1^k)$. Since we can compute $\mathrm{bin}(n)$ from $\mathrm{tal}(n)$ in polynomial time, it follows that any tree which is p-time bounded in tally is also p-time bounded in binary. Note that any tree $T\subset\{0,1\}^{<\omega}$ is p-time bounded, so that a tree may be p-time bounded without being p-time. One additional observation is worth making at this point. If T is a p-time bounded in tally, then there will also be a p-time function q such that, for any $\tau=(n_0,\ldots,n_k)\in T$, $|\mathrm{tal}(\tau)|\leqslant q(1^k)$. To see this, note that τ con-

sists of the strings $\text{tal}(n_i)$ for $i \leqslant k$, separated by commas and with parentheses at the beginning and end. Thus

$$|\tau| = 2 + k + |n_0| + \cdots + |n_k| \leqslant 2 + k + p(1^0) + \cdots + p(1^k).$$

Thus we can define a p-time bound $q(1^k) = 2 + k + p(1^0) + \cdots + p(1^k)$, which is clearly p-time computable. The same observation holds for p-time bounded in binary.

Now suppose that T is p-time bounded in tally and that $\text{Tal}(T)$ is p-time. This implies that there are at most $p(1^k)$ possible choices for $\text{tal}(n_k)$, that is, the strings 1^e for $e < p(1^k)$. To compute $h(\sigma)$, where $\sigma = (\text{tal}(n_0), \ldots, \text{tal}(n_{k-1}))$, we simply use the p-time algorithm for membership in $\text{tal}(T)$ to test whether $\sigma * \text{tal}(i) \in \text{tal}(T)$ for all $i \leqslant p(1^k)$ and compile the list $(\text{tal}(i_1), \ldots, \text{tal}(i_d)) = h(\sigma)$ of all $\text{tal}(i)$ such that $\sigma * \text{tal}(i) \in T$. This shows that the function h is p-time in tally. It then follows by the discussion above that g and f are also p-time in tally. Hence in the tally representation, any p-time bounded, p-time tree is also locally p-time.

Let us say that a tree T is *highly p-time in binary* if T is p-time, locally p-time and also p-time bounded in binary. Similarly, T is highly p-time in tally if T is p-time, locally p-time and also p-time bounded in tally. Then we have shown that in tally, p-time plus p-time bounded implies highly p-time. On the other hand, we have also seen that these notions are distinct for the binary representation.

Similar definitions can be given for other notions of complexity. Our next theorem shows that any Π^0_1-class can be realized as the set of infinite paths through a p-time tree.

Theorem 4.1. *Let T be a recursive tree. Then there is a polynomial time tree P such that $[T] = [P]$. Furthermore, if T is recursively bounded, then P is also recursively bounded and if T is p-time bounded, then P is also p-time bounded.*

Proof. The same argument works for the binary and for the tally representation. We will give the binary argument for the first part and the tally argument for the second part, since these are the stronger results. Let ϕ be a recursive function from $\omega^{<\omega}$ into $\{0, 1\}$ such that $\sigma \in \text{bin}(T) \Leftrightarrow \phi(\sigma) = 1$. Let ϕ^s denote the partial recursive function which results by computing ϕ for exactly s steps on any input and let T^s be the sth approximation to T, given by

$$\sigma \in T^s \Leftrightarrow \phi^s(\text{bin}(\sigma)) = 1 \text{ or is undefined.}$$

Thus $T^0 \supset T^1 \supset \cdots$ and, for any σ, $\sigma \in T \Leftrightarrow (\forall s)(\sigma \in T^s)$.

Now define the p-time tree P by letting

$$\sigma \in P \Leftrightarrow (\forall \tau < \sigma)\tau \in T^{|\text{bin}(\sigma)|}.$$

Note that P is a p-time tree in binary since to compute whether $\tau \in T^{|\text{bin}(\sigma)|}$ requires $|\text{bin}(\sigma)|$ steps for all τ so that to compute whether $\sigma \in P$ requires roughly $|\text{bin}(\sigma)|\,(|\text{bin}(\sigma)| + 1)$ steps.

It follows from the definition of P that $T \subset P$, so that $[T] \subset [P]$. Now suppose that

$x \notin [T]$. Then there is some initial segment $\tau = x \upharpoonright n$ which is not in T. This means that, for some s, $\tau \notin T^s$. Since the sequence T^s is decreasing, we may assume that $s > n$. Now let $\sigma = x \upharpoonright s$, so that $|\text{bin}(\sigma)| \geqslant s$. It follows from the definition of P that $\sigma \notin P$. This implies that $x \notin [P]$. Thus $[T] = [P]$.

Now suppose that T is recursively bounded in tally and let p be the recursive function which computes, for each k, an upper bound $p(1^k)$ (in tally) for the possible value of n_k for any node $\sigma = (n_0, \ldots, n_k) \in T$.

Suppose first that p is actually p-time. Then we can recursively define a tree Q such that $T \subset Q \subset P$ by putting $\sigma = (n_0, \ldots, n_k) \in Q$ if and only if $\sigma \in P$ and, for all $i \leqslant k$, $n_i \leqslant p(1^i)$. It is clear that $[Q] = [T]$ and that Q is p-time since P and p are p-time.

Finally, suppose only that p is recursive and let p^s be the usual result of computing p for s steps. Once again we can define a highly recursive tree Q such that $T \subset Q \subset P$ by putting $\sigma = (n_0, \ldots, n_k) \in Q$ if and only if $\sigma \in P$ and, for all $i \leqslant k$, either $p^k(1^i)$ is undefined or $n_i \leqslant p^k(1^i)$. Then again it is easy to check that Q is p-time in binary and that $[Q] = [T]$. □

Next we consider conditions which might force the tree T to have a recursive path, or possibly a p-time (exponential time, etc.) path. For any tree T, let $\text{Ext}(T)$ be the set of infinitely extendible nodes of T, that is,

$$\text{Ext}(T) = \{\sigma \in T : (\forall n > |\sigma|)(\exists \tau)(|\tau| = n \wedge \tau \in T \wedge \sigma \prec \tau).$$

Then, for any recursive tree T, $\text{Ext}(T)$ is a tree, although not necessarily a recursive tree. The following well-known result is basic to the subject, see Jockusch and Soare (1972a).

Theorem 4.2.

(a) *For any tree T, if $\text{Ext}(T)$ is recursive, then $[T]$ contains a recursive infinite member.*

(b) *If T is a highly recursive tree and $[T]$ is finite, then every member of $[T]$ is recursive.* □

Corollary 4.3.

(a) *For any recursively closed game $G(X)$,* if every partial strategy can be extended to a full winning strategy, *then there is a recursive winning strategy for $G(X)$.*

(b) *For any recursively closed game $G(X)$, if there are only finitely many winning strategies, then every winning strategy is recursive.*

Proof. Note first that any winning strategy for Player I must win some finite truncated game (by compactness). Thus if Player I has a winning strategy, then he has a recursive winning strategy. Now if Player II has the winning strategy, we just apply Theorem 4.2 to the recursively bounded Π^0_1-class $WS(T)$. □

Next we consider p-time versions of the winning strategies problem for infinite games with perfect information. We know from Theorem 3.4 that the set of winning strategies of a recursive game can represent an arbitrary Π^0_1-class and that, in particular, a recursive game need not have a recursive winning strategy. The next result shows that the same can be said for p-time closed games. Now recall that if a tree T is p-time in binary, then T is also p-time in tally and that if T is p-time bounded in tally, then T is also p-time bounded in binary. Thus we will say that a set $Y \subseteq \{0,1\}^\omega$ and the associated game $G(Y)$ is *p-time closed* if $Y = [T]$ for some tree T which is p-time in binary and we will say that Y and $G(Y)$ are *p-time bounded* if the tree T is p-time bounded in tally.

Theorem 4.4. *For any recursively closed game $G(X)$, there is a p-time closed game $G(Y)$ with the same set of winning strategies. Furthermore, if $G(X)$ is p-time bounded, then $G(Y)$ is also p-time bounded.*

Proof. The game is defined by a highly recursive tree T, so that Player II wins a play x of the game if and only if $x \in X = [T]$. Let the tree P with $[T] = [P]$ be given by Theorem 4.2 so that P is p-time in binary, is highly recursive and is p-time bounded if T is p-time bounded. Then it is clear that any strategy for Player II is winning for $G(X)$ if and only if it is winning for $G(Y)$. □

Next we show that the obvious p-time analogs of Theorem 4.2 and Corollary 4.3 fail for p-time trees and games.

Theorem 4.5.

(a) *For any recursive $x \in \{0,1\}^\omega$, there is a tree T which is p-time in binary and in tally and such that $[T] = \{x\}$.*

(b) *For any recursive $y \in \{0,1\}^\omega$, there is a p-time closed game $G(Y)$ such that Player II has exactly one winning strategy for $G(Y)$ and that strategy is Turing equivalent to y.*

Proof. (a) This follows from Theorem 4.1, since for $x \in \{0,1\}^\omega$, the tree $T = \{(x(0), \ldots, x(n-1) : n < \omega\}$ is recursive.

(b) Given a fixed y, define the Π^0_1-class X_y by $X_y = \{x(0), y(0), x(1), y(1), \ldots) : y = (y(0), y(1), \ldots)\}$. It is clear that Player II has a unique winning strategy θ for the game $G(X_y)$ which is given by $\theta(x(0), y(0), \ldots, y(n-1), x(n)) = y(n)$ and is Turing equivalent to y. Now let the p-time closed game $G(Y)$ be given by Theorem 4.4 so that $G(Y)$ and $G(X_y)$ have the same set of winning strategies. □

Theorem 4.5 shows that even if a p-time bounded p-time tree has a unique infinite path Π, Π may not be p-time and similarly if a highly p-time game has a unique winning strategy, that strategy need not be p-time. However, there are some natural conditions which we can put on T which will ensure that in such situations we can

at least get double exponential time paths or winning strategies and in some cases actually guarantee the existence of p-time paths or winning strategies.

Before we can state such a result, we need to introduce another notion of feasibility. A function f is said to be *non-deterministic p-time* (NP) if there is a finite alphabet Σ, a polynomial p, a p-time relation R and a p-time function g such that, for any σ and τ,

$$f(\sigma)=\tau \Leftrightarrow (\exists \varrho \in \Sigma^{p(|\sigma|)})\{R(\varrho,\sigma) \text{ and } g(\varrho,\sigma)=\tau\}.$$

Theorem 4.6.

(a) *Let* $\mathrm{Ext}(T)$ *be a locally p-time tree in tally (resp. binary) and let* $[T]$ *be non-empty. Then* $[T]$ *contains an infinite path which is double-exponential-time computable in tally (resp. binary). Furthermore, if* $\mathrm{Ext}(T)$ *is locally p-time in tally (resp. binary) and* $[T]$ *is finite, then every element of* $[T]$ *is computable in double exponential time in tally (resp. binary).*

(b) *Let* $\mathrm{Ext}(T)$ *be a locally p-time tree in tally (resp. binary) and let* $[T]$ *be non-empty. Moreover, assume that there is a linear time function h such that for all* $\sigma=(n_0,\ldots,n_k)\in T$, $h(b(\sigma))$ *lists all* $b(n)$ *such that* $(n_0,\ldots,n_k,n)\in T$, *where* $b(\cdot)=\mathrm{tal}(\cdot)$ *if T is p-time in tally and* $b(\cdot)=\mathrm{bin}(\cdot)$ *if T is p-time in binary. Then* $[T]$ *contains an infinite path which is exponential-time computable in tally (resp. binary). Furthermore, if* $[T]$ *is finite, then every element of* $[T]$ *is computable in exponential time in tally (resp. binary).*

(c) *If* $\mathrm{Ext}(T)$ *is a highly p-time tree in tally (resp. binary) and* $[T]$ *is non-empty, then* $[T]$ *contains an infinite path which is p-time in tally (resp. binary). Furthermore, if* $[T]$ *is finite, then every element of* $[T]$ *is p-time in tally (resp. binary).*

(d) *If* $\mathrm{Ext}(T)$ *is a p-time bounded, p-time tree in binary and* $[T]$ *is non-empty, then* $[T]$ *contains an infinite path which is EXPTIME in binary. Furthermore, if* $[T]$ *is finite, then every element of* $[T]$ *is NP in binary.*

Proof. To simplify the discussion, we will assume in all cases that $T=\mathrm{Ext}(T)$, that is, that T has no dead ends. Thus the conditions set out for $\mathrm{Ext}(T)$ will become the conditions for T.

(a) We give the proof for the binary representation of T. The proof for the tally representation is exactly the same except for replacing $\mathrm{bin}(\ldots)$ with $\mathrm{tal}(\ldots)$ at appropriate locations throughout. Let h be the p-time function such that for all $\sigma=(n_0,\ldots,n_k)\in T$, $h(\mathrm{bin}(\sigma))$ lists all $\mathrm{bin}(n)$ such that $(n_0,\ldots,n_k,n)\in T$. Then we can recursively define the p-time path x through T by letting $x(k)$ be the number n such that $\mathrm{bin}(n)$ is the first entry of $h(\mathrm{bin}(x\upharpoonright k))$. It remains to be checked that the computation of $\mathrm{bin}(x(n))$ from 1^n can be done in double exponential time. Let c be a number such that $h(\tau)$ can be computed from τ in time bounded by $|\tau|^{c-1}$ for all $\tau\in\mathrm{Bin}(\mathrm{Ext}(T))$ with $|\tau|\geqslant 2$. For each k, let $\tau_k=(\mathrm{bin}(x(0),\ldots,\mathrm{bin}(x(k-1))$. Then $\tau_0=\emptyset$ and, for each $k>0$, τ_{k+1} can be computed from τ_k in time bounded by $|\tau_k|^c$. (Just start the computation of $h(\tau_k)$, stop it as soon as you have the first element

$\varrho = \mathrm{bin}(x(k))$ in the list and then append ϱ to the end of τ_k.) Thus in particular $|\tau_{k+1}| \leqslant |\tau_k|^c$ for all $k>0$. Now choose c large enough so that $|\tau_1| \leqslant 2^c$. It then follows by induction that, for all k, $|\tau_k| \leqslant 2^{c^k}$. It follows that the computation of τ_{k+1} from τ_k can be done in time bounded by $(2^{c^k})^{c-1} \leqslant 2^{c^{k+1}}$. Thus the entire computation of $\tau_n = (\mathrm{bin}(x(0)), \ldots, \mathrm{bin}(x(n)))$ from 1^n takes time bounded by

$$\sum_{k<n} 2^{c^{k+1}} < 2^{c^n+1} < 2^{(c+1)^n},$$

which shows that x is computable in double exponential time in binary.

Now suppose that $[T]$ is finite and let $x \in [T]$. By restricting T to the extensions of $x \upharpoonright n$ for sufficiently large n, we may assume that x is the unique infinite path through T. The result now follows immediately from (a).

(b) The proof is essentially the same as the proof of (a). Again we shall only give the proof in the case that T is p-time in binary. The point is that if h is linear time, then it follows that for all $k \geqslant 1$, τ_{k+1} can be computed from τ_k in time $c \cdot |\tau_k|$ for some fixed constant c. If we pick c so that $c \geqslant |\tau_0|$, then it is easy to prove by induction that $|\tau_k| \leqslant c^{k+1}$ for all $k \geqslant 0$. Thus the entire computation of $\tau_n = (\mathrm{bin}(x(0)), \ldots, \mathrm{bin}(x(n)))$ from 1^n takes time bounded by

$$\sum_{k<n} c^{k+2} < (n+1)c^{n+1} < c^{2n+2},$$

which shows that x is computable in exponential time in binary.

Now if $[T]$ is finite and $x \in [T]$, then again by restricting T to the extensions of $x \upharpoonright n$ for sufficiently large n, we may assume that x is the unique infinite path through T. Then by the above argument it follows that $x \in DEXT$.

(c) The proof will be a minor modification of the proof of (a) above. Again the proofs are the same for tally and for binary so we will just give a binary version. Let q be a p-time function such that for any $\tau = (\mathrm{bin}(n_0), \ldots, \mathrm{bin}(n_k)) \in \mathrm{bin}(T)$, $|\tau| \leqslant q(1^k)$. Since $|1^k| = k$, it follows that for some constant b and all $k>1$, we have $|\tau_k| \leqslant k^b$. Let c be a number such that $h(\tau)$ can be computed from τ in time bounded by $|\tau|^{c-1}$ for all $\tau \in \mathrm{Bin}(\mathrm{Ext}(T))$ with $|\tau| > 1$. Then the computation of τ_{k+1} from τ_k can be done in time bounded by $|\tau_k|^c \leqslant k^{bc}$ for all $k>1$. Now let a be large enough so that $a > bc$ and also large enough so that both τ_0 and τ_1 can be computed in time bounded by a. Then the entire computation of $\tau_n = (\mathrm{bin}(x(0)), \ldots, \mathrm{bin}(x(n)))$ from 1^n takes time bounded by

$$a + 2^a + 3^a + \cdots + (n-1)^a \leqslant \sum_{k<n^a} k \leqslant n^{2a},$$

which shows that x is computable in p-time in binary.

If we assume further that $[T]$ is finite, then the same argument as given in (a) and (b) above shows that every element of $[T]$ is p-time computable.

(d) As in (c) we may assume that if $\tau = (\mathrm{bin}(n_0), \ldots, n_k) \in T$ and $k>1$, then $|\tau| \leqslant k^b$ so that in particular $n_k \leqslant k^b$. In this case, we are not assuming that T is locally p-time, so that we need a different algorithm for producing an infinite path

x in $[T]$. We will define $x(k)$ recursively by making $x(k)$ be the least number n such that $(x(0),\dots,x(k-1),n)\in T$. This means that we may have to check whether $(x(0),\dots,x(k-1),x)\in T$ for all x with $|\mathrm{bin}(x)|\leqslant k^b$. This is where the binary representation differs from the p-time representation, because there will now be 2^{k^b} different strings to check. Each check will require time at most $(k^b)^c$, so that the computation of $\mathrm{bin}(x(k))$ from $(\mathrm{bin}(x(0)),\dots,\mathrm{bin}(x(k-1))$ will require time less than $2^{k^{bc+b}}$ for $k>1$. Now let a be large enough so that $a\geqslant bc+b$ and also large enough so that $\mathrm{bin}(x(0))$ and $\mathrm{bin}(x(1))$ can be computed in time $\leqslant a$. Then the entire computation of $\tau_n=(\mathrm{bin}(x(0),\dots,\mathrm{bin}(x(n)))$ from 1^n takes time bounded by

$$a+2^{2^a}+2^{3^a}+\cdots+2^{(n-1)^a}\leqslant\sum_{k<n^a}2^k\leqslant 2^{n^{2a}},$$

which shows that $x\in EXPTIME$.

If we assume further that $[T]$ is finite, then the same argument as given in (a) above shows that every element of $[T]$ is *EXPTIME*. However, it is easy to show that the infinite paths through T are actually *NP*-computable.

As above, we may assume that T has no dead ends and has a unique infinite path x. Thus for any k, $x(0),x(1),\dots,x(k))$ is the unique infinite path in T with $k+1$ entries. Furthermore, since T is p-time bounded, we know as above that $|(\mathrm{bin}(x(0)),\dots,\mathrm{bin}(x(k)))|\leqslant k^b$ for some fixed b. Thus to compute $x(k)$ nondeterministically, we simply guess a string $\sigma=(\mathrm{bin}(n_0),\dots,\mathrm{bin}(n_k))$ of length $\leqslant k^b$ and then use the p-time algorithm for T to test whether $\sigma\in T$. When the answer is yes, we read the value of $x(k)$ from the end of σ. Since there is only one possible correct guess for σ, this procedure will compute $x(k)$. □

Next we shall give two examples to show that the bounds given in parts (a) and (b) of Theorem 4.6 cannot be improved. Consider the following.

Example 1. A locally p-time tree T with a unique infinite path x such that $T=EXT(T)$ and x is double exponential time.

Let $x(n)=2^{2^n}$ for all n and let the tree T consist of all initial segments of x. Then $(n_0,\dots,n_k)\in T$ if and only if $n_0=1$ and, for all $i<k$, $n_{i+1}=n_i^2$. It is clear that both $\mathrm{tal}(T)$ and $\mathrm{bin}(T)$ are p-time. Furthermore, for any $\sigma=(n_0,\dots,n_k)\in T$, we have $h(\sigma)=n_k^2$, so that T is locally p-time in both binary and tally.

Example 2. A locally p-time tree T with a unique infinite path x such that $T=EXT(T)$, there is a linear time function h such that for all $\sigma=(n_0,\dots,n_k)\in T$, $h(b(\sigma))$ lists all $b(n)$ such that $(n_0,\dots,n_k,n)\in T$ where $b(\cdot)=\mathrm{tal}(\cdot)$ if T is p-time in tally and $b(\cdot)=\mathrm{bin}(\cdot)$ if T is p-time in binary, and x is exponential time but not p-time.

Let $x(n)=2^n$ for all n and let the tree T consist of all initial segments of x. Then $(n_0,\dots,n_k)\in T$ if and only if $n_0=1$ and, for all $i<k$, $n_{i+1}=2_{n_i}$. It is clear that both

tal(T) and bin(T) are p-time and that the function h is linear time in both cases, so that T is locally p-time in both binary and tally.

To apply Theorem 4.6 to p-time games, we need to consider the complexity of the definition of $WS(T)$ from T.

Lemma 4.7.

(a) *If the tree T is p-time in binary (resp. tally), then $WS(T)=[Q]$, where Q is also a p-time tree in binary (resp. tally).*

(b) *If the tree T is locally p-time in binary (resp. tally), then $WS(T)=[Q]$, where Q is also a locally p-time tree in binary (resp. tally).*

(c) *If the tree T is p-time bounded in binary (resp. tally), then $WS(T)=[Q]$, where Q is also a p-time bounded tree in binary (resp. tally).*

Proof. We shall give details in the case where T is a $\{0,1\}$-tree, that is, where $T\subset\{0,1\}^{<\omega}$. Then we will outline the proof for the general case. Now let T be a p-time $\{0,1\}$-tree and let Q be the tree defined in the proof of Theorem 3.2 with $WS(T)=[Q]$. Then the binary and tally representations of T are essentially the same and T is automatically p-time bounded and locally p-time. It follows from the definition of Q that Q is also a $\{0,1\}$-tree and is therefore p-time bounded and locally p-time. Thus we need only show that Q is p-time. Given a string $\sigma=(e_0,\ldots,e_{n-1})$, the following is a procedure for testing whether $\sigma\in Q$. Recall that $\tau_0,\tau_1,\ldots$ is an enumeration of the non-empty strings in $\{0,1\}^*$ in order, first by length and then lexicographically. To make this more computational, recall that a binary number is simply either 0 or a string ending in a 1. Thus we can view τ_k as the string such that $\tau_k{}^\frown 1=\mathrm{bin}(k+2)$. This demonstrates that the map between 1^k and τ_k is p-time and that, in fact, we can compute the list $\tau_0,\tau_1,\ldots,\tau_n$ from 1^n in polynomial time. Now we can view $\tau_0,\tau_1,\ldots$ as a listing of all possible partial plays of Player I. The procedure is to apply the partial strategy θ_σ to each partial play τ_i with $i\leqslant k$ to produce the play $\theta_\sigma(\tau_i)$. Recall that if $\tau_i=(x(0)\ldots x(t))$, then $\theta_\sigma(\tau_i)=(x(0),y(0),\ldots,x(t),y(t))$, where for $j\leqslant t$, $y(j)=e_{l_j}$ if $\tau_{l_j}=(x(0),\ldots,x(j))$. This recursive definition shows that we can apply the strategy θ_σ to the list of partial plays $\tau_0,\ldots,\tau_{n-1}$ of Player I to produce in p-time from σ a list of n partial plays of the game. Finally, we check to see that every one of these plays is in the tree T. Since T is a p-time tree, we can test each play in the list in p-time in its length and therefore we can test the whole list in p-time in the length of the list. Since the list is computed from σ in p-time, this shows that the entire procedure can be done in p-time in σ. This shows that Q is a p-time tree.

Now suppose that T is simply a p-time tree in tally which is merely contained in $\omega^{<\omega}$. We will indicate how to define the Π^0_1-class $WS(T)$ and define the tree Q with $WS(T)=[Q]$. As in the argument above, the key idea is that a sequence $\sigma=(\mathrm{tal}(e_0),\ldots,\mathrm{tal}(e_n))$ will represent the partial strategy of responses of Player II to the possible partial plays $\tau_0,\ldots,\tau_n$ of Player I in the game $G(T)$. Now the possible partial plays $\tau_0,\tau_1,\ldots$ of Player I will be sequences of tally numbers, defined as

follows. For each n, let $\text{bin}(n+1) = 0^{n_1}10^{n_2}1\cdots 10^{n_k}1$, where k is just the number of occurrences of 1 in $\text{bin}(n+1)$. Then $\tau_n = (\text{tal}(n_1), \ldots, \text{tal}(n_k))$. Observe that any initial segment $(\text{tal}(n_1), \ldots, \text{tal}(n_i))$ of τ_n is τ_m, where $\text{bin}(m+1) = 0^{n_1}1\cdots 0^{n_i}1$ is an initial segment of $\text{bin}(n+1)$, so that $m < n$ and τ_m is the list of plays up to τ_n. Note here that $\text{bin}(n)$ can be computed from 1^n in p-time by the usual method of repeatedly dividing by 2 and taking the remainder, so that we can compute from 1^n the entire list $\tau_0, \ldots, \tau_n$. Then as above a path $\sigma = (\text{tal}(e_0), \ldots, \text{tal}(e_n))$ represents the partial strategy of responses to the partial plays $\tau_0, \ldots, \tau_n$. Now from input σ, we can compute the list of partial plays $\theta_\sigma(\tau_i)$ of the game $G(T)$ and then use the p-time algorithm for T to check whether each partial play is in T. Then $\sigma \in Q$ if and only if each of the partial plays is in T. This shows that Q is p-time.

Next suppose that T is locally p-time and let $\sigma = (\text{tal}(e_0), \ldots, \text{tal}(e_n))$. To show that Q is locally p-time, we need to be able to compute from σ in p-time a lists of the $\text{tal}(e)$ such that $\sigma * (\text{tal}(e)) \in Q$. Now $\text{tal}(e)$ must represent the response to the partial play τ_{n+1} of Player I. Thus we first compute $\text{tal}(n)$ from σ and then compute $\tau_{n+1} = (x(0), \ldots, x(k))$. Next we compute for each $i < k$, the number m_i such that $\tau_{m_i} = (x(0), \ldots, x(i))$ and then compute the partial play $\pi = (x(0), y(0), \ldots, x(k-1), y(k-1), x(k))$, where $y(i) = \text{tal}(e_{m_i})$ for each i. Then since $\text{tal}(e)$ represents the next move of Player II in the game $G(T)$, we see that $\sigma * (\text{tal}(e)) \in Q$ if and only if $\pi * (\text{tal}(e)) \in T$. Since T is locally p-time, we can now compute from π the list of $\text{tal}(e)$ such that $\sigma * (\text{tal}(e)) \in Q$. This shows that Q is also locally p-time.

Finally, suppose that T is p-time bounded and let $\sigma = (\text{tal}(e_0), \ldots, \text{tal}(e_n)) \in Q$. To show that Q is p-time bounded, we must be able to compute from 1^n in p-time an upper bound $q(1^n)$ for e_n. Now suppose that $\tau_n = (x(0), \ldots, x(k))$. It follows that $\text{tal}(e_n) = y(k)$ for some partial play $(x(0), y(0), \ldots, x(k), y(k)) \in T$ of the game $G(T)$. Since T is p-time bounded, there is a p-time function p such that $|y(k)| \leqslant p(\text{tal}(2k+1))$. It follows that $e_n = |\text{tal}(e_n)| \leqslant p(\text{tal}(2k+1))$. Recall from the definition of τ_n above that k is simply the number of 1's in $\text{bin}(n)$. Thus from input 1^n, we can compute 1^k and then compute $q(1^n) = p(1^k)$. This shows that Q is also p-time bounded.

If T is a p-time in binary, the argument is similar. The list $\tau_0, \tau_1, \ldots$ of possible partial plays of Player I will be defined by $\tau_n = (\text{bin}(n_0), \ldots, \text{bin}(n_k))$, where $n = 0^{n_0}1\cdots 10^{n_k}1$. □

Theorem 4.8. *Let $G(T)$ be a p-time Gale–Stewart game for which Player II has a winning strategy and such that every partial (winning) strategy of Player II has an extension to an infinite winning strategy.*

(a) *Let T be locally p-time in binary (resp. tally). Then Player II has a winning strategy for the game $G(T)$ which is computable in double exponential time in binary (resp. tally). If there are only finitely many different winning strategies for Player II in the game $G(T)$, then every winning strategy for Player II in the game $G(T)$ is computable in double exponential time.*

(b) *Let T be highly p-time in binary (resp. tally). Then Player II has a winning*

strategy for the game $G(T)$ which is computable in p-time in binary (resp. tally). If there are only finitely many different winning strategies for Player II in the game $G(T)$, then every winning strategy for Player II in the game $G(T)$ is computable in p-time.

(c) *Let T be p-time bounded and p-time in binary. Then Player II has a winning strategy for the game $G(T)$ which is computable in EXPTIME in binary. If there are only finitely many different winning strategies for Player II in the game $G(T)$, then every winning strategy for Player II in the game $G(T)$ is computable in non-deterministic polynomial time (NP time).*

Proof. Let Q be the tree such that $WS(T) = [Q]$. Since T is p-time, it follows from Lemma 4.7(a) that Q is p-time. Since Player II has a winning strategy for the game $G(T)$, it follows that $WS(T)$ is non-empty. Since every partial strategy for Player II has an extension to an infinite strategy, it follows that Q has no dead ends, that is, $\mathrm{Ext}(Q) = Q$.

(a) Since T is locally p-time, it follows from Lemma 4.7(b) that Q is also locally p-time. The result now follows from Theorem 4.6(a).

(b) Since T is highly p-time, that is, locally p-time and p-time bounded, it follows from Lemma 4.7(b) and (c) that Q is also highly p-time. The result now follows from Theorem 4.6(c).

(c) Since T is p-time bounded in binary, it follows from Lemma 4.7(c) that Q is also p-time bounded and p-time in binary. The result now follows from Theorem 4.6(d). □

Acknowledgements

The second author was partially supported by NSF grant DMS-90-06413. Both authors were partially supported by the Army Research Office through the Mathematical Sciences Institute of Cornell University.

References

D. Cenzer and J. Remmel, Polynomial-time versus recursive models, Ann. Pure Appl. Logic 54 (1991) 17–58.

D. Cenzer and J. Remmel, Polynomial-time Abelian groups, Ann. Pure Appl. Logic 56 (1992a) 313–363.

D. Cenzer and J. Remmel, Feasible graphs and colorings. Methods of Logic in Computer Science (1992b) (to appear).

D. Cenzer and J. Remmel, Π^0_1 classes in mathematics (1992c) (to appear).

D. Gale and F.M. Stewart, Infinite games with perfect information, Ann. Math. Studies 28 (1953) 245–266.

M. Garey and O. Johnson, Computers and Intractability (Freeman, New York, 1978).

C.G. Jockusch and R.I. Soare, Π^0_1-classes and degrees of theories, Trans. Am. Math. Soc. 173 (1972a) 33–56.

C.G. Jockusch and R.I. Soare, Degrees of members of Π^0_1-classes, Pacific J. Math. 40 (1972b) 605–616.

J.E. Hopcroft and J.D. Ullman, Formal Languages and Their Relations to Automata (Addison Wesley, 1969).

A. Kucera, An alternative, priority free, solution to Post's problem, Lecture Notes Comput. Sci. 233 (1986) 493–500.

A. Manaster and J. Rosenstein, Effective matchmaking, Proc. London Math. Soc. 25 (1972) 615–644.

G. Metakides and A. Nerode, Effective content of field theory, Ann. Math. Logic 17 (1979) 289–320.

Y.N. Moschovakis, Descriptive Set Theory, Studies in Logic, Vol. 100 (North-Holland, Amsterdam, 1980).

J.B. Remmel, Graph coloring and recursively bounded Π^0_1-classes, Ann. Pure Appl. Logic 32 (1986) 185–194.

H.J. Rogers, Theory of Recursive Functions and Effective Computability (McGraw-Hill, New York, 1967).

J. von Neumann and O. Morgenstern, Theory of Games and Economic Behavior (Princeton University Press, 1947).