How to correctly prune tropical trees

Jean-Vincent Loddo and Luca Saiu

Laboratoire d'Informatique de l'Université Paris Nord - UMR 7030
Université Paris 13 - CNRS
99, avenue Jean-Baptiste Clément - F-93430 Villetaneuse
{loddo,saiu}@lipn.univ-paris13.fr

Abstract. We present *tropical games*, a generalization of combinatorial *min-max* games based on tropical algebras. Our model breaks the traditional symmetry of rational zero-sum games where players have exactly opposed goals (*min* vs. *max*), is more widely applicable than *min-max* and also supports a form of pruning, despite it being less effective than α-β. Actually, *min-max* games may be seen as particular cases where both the game and its dual are tropical: when the dual of a tropical game is also tropical, the power of α-β is completely recovered. We formally develop the model and prove that the tropical pruning strategy is correct, then conclude by showing how the problem of approximated parsing can be modeled as a tropical game, profiting from pruning.

Key words: combinatorial game, search, alpha-beta pruning, rational game, tropical algebra, tropical game, term, rewriting, logic, parsing

1 Introduction

We are all familiar with games such as Chess or Checkers. Such games are purely *rational* as they do not involve any element of chance; they are also *zero-sum*, as the players' interests are dual: what one "wins", the other "loses" — which is the origin of the *min-max* evaluation mechanism. The two fundamental questions to be asked in a rational game are *"Who will win?"* and *"How much will she win?"*. Answering such questions involves *searching for a strategy* trough a (typically large) *game tree*. Some optimized search techniques were developed, which in the case of combinatorial two-player games include the α-β *pruning* technique [1,2]. α-β is not an approximated algorithm: its correctness relies on the mutual distributive properties of *min* and *max*. In this work we explore the implications of assuming only *one* player to be rational, breaking the symmetry of the traditional "double-sided" rationality. Quite unsurprisingly our *tropical α-pruning* depends on just *one* distributive property, a requirement satisfied by *tropical algebras* (Section 3).

Following the style introduced by [3] and [4], we will distinguish two aspects of two-player combinatorial games: a first one that we call *syntactic*, consisting

This work is partially financed by Marie Curie action n. 29849 Websicola and ANR-06-JCJC-0122.
The final publication of this paper is available at www.springerlink.com.

in a description of the possible game positions and the valid moves leading from a position to another; the game syntax is the formal equivalent of the intuitive notion of the "game rules". By contrast the *semantic* aspect is concerned about the interpretation of the game according to the interests of the players, and ultimately about the answer to the two fundamental questions above. Our semantics will be based on tropical algebras, and as a consequence our technique is widely applicable, relying as it does only on their comparatively weak hypotheses.

We formally define tropical α-pruning and prove its soundness, as our main contribution (Section 4). A further contribution consists in our formalization of game evaluation and tropical (and α-β) cuts as a small-step semantics, so that proofs can reuse the results of term-rewriting theory.

Actually, our soundness result subsumes other works proving α-β's soundness over distributive lattices such as [4] and (later) [5], since distributive lattices are bi-tropical structures (Definition 8).

We conclude by proposing the algorithm design style *Choose-How-To-Divide and Conquer* meant for attacking even apparently unrelated search problems as tropical games; we develop approximated parsing as one such problem by showing how it profits from α-pruning (Section 5).

2 Combinatorial game syntax and semantics

2.1 Syntax

We speak about "syntax", hinting at formal grammars, in that some initial game positions are given, together with some "rule" allowing to derive successive positions from those: in this way a game can be seen as the tree of all the possibilities of playing it — the tree of all the possible matches.

Definition 1 (Syntax). *A game syntax or arena is a triple $S = (\mathbb{P}, \lambda, succ)$, where:*

- *$\mathbb{P}$ is the set of all game positions.*
- *the* turn function *$\lambda : \mathbb{P} \to \{\mathcal{P}, \mathcal{O}\}$, says whose turn it is: $\mathcal{P}$ for "player" or $\mathcal{O}$ for "opponent".*
- *the* successor function *$succ$, taking a game position and returning all the positions reachable with valid moves from there; $succ : \mathbb{P} \to \mathbb{P}^*$.*

Given $S = (\mathbb{P}, \lambda, succ)$, we define:

- *the set of* terminal positions *$\mathbb{P}_T = \{\pi \in \mathbb{P} \mid succ(\pi) = \langle\rangle\}$.*
- *the* dual arena *$S^\perp = (\mathbb{P}, \lambda^\perp, succ)$, of course with $\lambda^\perp : \mathbb{P} \to \{\mathcal{P}, \mathcal{O}\}$, where for any $\pi \in \mathbb{P}$ we have $\lambda^\perp(\pi) \neq \lambda(\pi)$.*
- *the* move *relation is the binary version of the succ relation: for all $\pi, \pi' \in \mathbb{P}$, $move(\pi, \pi')$ iff $\pi' = \pi_i$ for some i, where $succ(\pi) = \langle \pi_1 ... \pi_n \rangle$.*

The arena is called alternate-turn *iff $move(\pi, \pi')$ implies $\lambda(\pi) \neq \lambda(\pi')$.*
If move is Nötherian we speak about Nötherian *or* finite *arena.*

Remark 1 (Alternate-turn arenas). It is possible to systematically make a game alternate-turn by "collapsing" all the *sequences of consecutive moves of the same player* into single moves.

One of the most important ideas in Game Theory is the *strategy*, containing a plan to win the game — a player saying to herself "if this happens I should do that, but if this other thing happens I should do that, and so on". It should be noticed that a strategy is only related to the syntactic part of a game, being independent, *per se*, from the game evaluation. In particular, a strategy may very well not be winning.

Definition 2 (Strategy). *Let $S = (\mathbb{P}, \lambda, succ)$ be an arena, and $\pi \in \mathbb{P}$ be a position. We define:*

- *the* reachable positions from π *as the right elements of the reflexive-transitive closure of the relation succ: $\pi{\downarrow} = succ^*(\pi)$;*
- *a* global strategy *σ, as a subset of the relation succ which is:*
  - *deterministic in $\mathcal{P}$ positions:*
    for all $\pi \in \mathbb{P}$ where $\lambda(\pi) = \mathcal{P}$, if $succ(\pi) = \langle \pi_1 ... \pi_n \rangle$ then $\sigma(\pi) = \langle \pi_i \rangle$, for some i such that $1 \leq i \leq n$.
  - *complete in $\mathcal{O}$ positions:*
    for all $\pi \in \mathbb{P}$ where $\lambda(\pi) = \mathcal{O}$, $\sigma(\pi) = succ(\pi)$.
- *a* strategy for the initial position *π is a global strategy for the restricted arena $S_\pi = (\pi{\downarrow}, \lambda|_{\pi\downarrow}, succ|_{\pi\downarrow})$, where we indicate with $f|_D$ the restriction of a function f to the set D.*

2.2 Semantics

Let us assume a finite game with syntax $S = (\mathbb{P}, \lambda, succ)$. Traditionally the two players have exactly opposed interests and we assume, by convention, that the player $\mathcal{P}$ will try to *minimize* the *payoff* of the final position while the opponent $\mathcal{O}$ will try to *maximize* it.

The ordinary way of *evaluating* such a finite game consists in labeling non-terminal nodes with the functions *min* and *max* (according to the turn), and terminal nodes with the *payoff* of the terminal position $p(\pi)$. Such values are then "propagated" back, applying the function at each node to its children's values. The final value at the root is called the *game value*: it says *who* wins and *how much*, supposing *both* players to be rational.

Hence, assuming $p : \mathbb{P}_T \to \mathbb{Z}$ in accord to the tradition, the game value $v_p : \mathbb{P} \to \mathbb{Z}$ could be simply defined as a function of the initial position:

$$v_p(\pi) = \begin{cases} p(\pi), & \pi \in \mathbb{P}_T \\ min_{i=1}^{n}\ v_p(\pi_i), & succ(\pi) = \langle \pi_1 ... \pi_n \rangle, \lambda(\pi) = \mathcal{P} \\ max_{i=1}^{n}\ v_p(\pi_i), & succ(\pi) = \langle \pi_1 ... \pi_n \rangle, \lambda(\pi) = \mathcal{O} \end{cases}$$

This classical definition has the obvious defect of only supporting the function *min* and *max*; often for resolving actual games the preferred structure is $\mathbb{Z}$, $\mathbb{Q}$ (possibly extended with $-\infty$ and $+\infty$), floating point numbers, or some sort of tuples containing such structures on which a topological order is defined. Hence, in order to be more general, let us define $\mathbb{U}$ to be any set closed over two associative binary operations $\oplus$ and $\odot$, where $\oplus$ will be associated to the player and $\odot$ to the opponent. Assuming $p : \mathbb{P} \to \mathbb{U}$, the definition above would become:

$$v_p(\pi) = \begin{cases} p(\pi), & \pi \in \mathbb{P}_T \\ \bigoplus_{i=1}^{n} v_p(\pi_i), & succ(\pi) = \langle \pi_1 ... \pi_n \rangle, \lambda(\pi) = \mathcal{P} \\ \bigodot_{i=1}^{n} v_p(\pi_i), & succ(\pi) = \langle \pi_1 ... \pi_n \rangle, \lambda(\pi) = \mathcal{O} \end{cases}$$

The extended v_p above is a step forward, but it still has the problem of only being well-defined on finite games. We solve this problem by abandoning the functional definition of v_p altogether, and giving a *small-step semantics* instead. Actually, this style will also be useful in Section 4.1 to prove the soundness of our pruning technique.

Remark 2 (Invariance under alternate-turn transformation). It is easy to see that the transformation hinted at in Remark 1 does not alter semantics, because of the two associative properties.

Definition 3 (Game). *A game is the triple $G = (S, \mathcal{A}, p)$, where $S = (\mathbb{P}, \lambda, succ)$ is the syntax, $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$ is an algebra with associative operations $\oplus$ and $\odot$, and where $p : \mathbb{P}_T \to \mathbb{U}$ is the payoff function.*

Sometimes we informally refer to syntactic or semantic properties as if they belonged to a game, for example by speaking about "Nötherian game" instead of "Game with Nötherian syntax".

Small-step operational semantics In the following, we assume a game $G = (S, \mathcal{A}, p)$, where $S = (\mathbb{P}, \lambda, succ)$ and $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$.
The configurations of our system consist of (ground) *terms* of G, recursively defined as: $Ter(G) = \mathbb{P} \uplus \mathbb{U} \uplus (\{\sum, \prod\} \times Ter(G)^+)$:

– positions in $\mathbb{P}$ indicate game positions still to be expanded (if not terminal) and evaluated (otherwise).
– values in $\mathbb{U}$ denote the value, already fully computed, of some sub-terms.
– a complex term such as $\sum\langle t_1 ... t_n\rangle$ or $\prod\langle t_1 ... t_n\rangle$ indicates a position at some state of its evaluation; $\sum$ or $\prod$ holding the turn information, and $t_1 ... t_n$ representing the game subterms from that state on.

It is crucial not to mistake *terms of* G, which represent partially expanded game trees, for *game positions*, which in practice will also tend to be structured symbolic terms, but can be considered atomic at a high level: the rewrite rules shown in the following work on $Ter(G)$, not on $\mathbb{P}$.

Syntactic conventions We use (possibly with subscripts or primes) π to indicate positions in $\mathbb{P}$, s and t for generic terms, v for values in $\mathbb{U}$, $\boldsymbol{t}$ and $\boldsymbol{z}$ for of terms in $Ter(G)$. Sequences are allowed to be empty, if not specified otherwise in a side condition. Just to make the notation more compact we will write $\sum \boldsymbol{t}$ instead of $(\sum, \boldsymbol{t})$ and $\prod \boldsymbol{t}$ for $(\prod, \boldsymbol{t})$. We write Λ instead of either $\sum$ or $\prod$, just to avoid duplicating otherwise identical rules. Sequences are written with no commas, and parentheses or brackets are used to group when needed.

$$[\text{Payoff}]\ \frac{\pi \in \mathbb{P}_T \qquad p(\pi) = v}{\pi \to v}$$

$$[\mathcal{P}\text{-expand}]\ \frac{succ(\pi) = \boldsymbol{t} \qquad \lambda(\pi) = \mathcal{P}}{\pi \to \sum \boldsymbol{t}}\ \#\boldsymbol{t} \geq 1$$

$$[\mathcal{O}\text{-expand}]\ \frac{succ(\pi) = \boldsymbol{t} \qquad \lambda(\pi) = \mathcal{O}}{\pi \to \prod \boldsymbol{t}}\ \#\boldsymbol{t} \geq 1$$

$$[\mathcal{P}\text{-reduce}]\ \frac{}{\sum \boldsymbol{t}\langle v_1\ v_2\rangle \boldsymbol{z} \to \sum \boldsymbol{t}\langle v\rangle \boldsymbol{z}}\ v_1 \oplus v_2 = v$$

$$[\mathcal{O}\text{-reduce}]\ \frac{}{\prod \boldsymbol{t}\langle v_1\ v_2\rangle \boldsymbol{z} \to \prod \boldsymbol{t}\langle v\rangle \boldsymbol{z}}\ v_1 \odot v_2 = v$$

$$[\text{Return}]\ \frac{}{\Lambda\langle v\rangle \to v}$$

$$[\text{Context}]\ \frac{t \to t'}{C[t] \to_c C[t']}\ \text{for all contexts } C$$

[Payoff] simply replaces a terminal position with its value in $\mathbb{U}$, by means of the payoff function. [$\mathcal{P}$-expand] and [$\mathcal{O}$-expand] expand a position, generating its successors and keeping track of the turn, which will be important at reduction time. [$\mathcal{P}$-reduce] and [$\mathcal{O}$-reduce] combine two values into one, using $\oplus$ for the player and $\odot$ for the opponent. Notice that these two rules are sources of non-determinism. [Return] unwraps a completely evaluated term containing a single value. [Context] allows to use the other rules within nested terms (also introducing non-determinism).

Notice that keeping the relation $\to_c$ distinct from $\to$ allows us, when needed, to see our semantics as a *term rewriting system* (TRS) [6].

Proposition 1. *$\to_c$ is strongly confluent.*

Proof. For the purposes of this proof, we consider the small-step semantics as a pure term-rewriting system, expressed in a slightly sugared notation. The system does *not* need to be conditional (CTRS), since all the rule premises can in fact be seen as structural constraints on syntactic constructors. $\oplus$ and $\odot$ should also be read as syntactic constructors, with their associative properties written as rewrite rules. What is a variable in the rules becomes a (syntactic) variable in the TRS; however, we will not exploit the full power of the formal system: reductions will only be applied to ground terms[1].

[1] We do not need the full power of unification: from a programming point of view, *pattern matching* as used in ML or Haskell is enough for our purposes.

Our TRS is trivially *left-* and *right-linear*, as no variable occurs more than once in each side of a rule. By showing that our system is also *strongly closed*, strong confluence follows by Huet's Lemma 3.2 in [7]: "If $\mathscr{R}$ is a left- and right-linear strongly closed term rewriting system, $\to_{\mathscr{R}}$ is strongly confluent".

In order to show that the system is strongly-closed, we have to show that for every critical pair s, t there exist s', t' such that $s \to^* t' \leftarrow^{\equiv} t$ and $t \to^* s' \leftarrow^{\equiv} s$ (as in [7] and [6]), where $\leftarrow^{\equiv}$ is the reflexive closure of $\leftarrow$.

The left-hand side of [$\mathcal{P}$-reduce] is $\sum \boldsymbol{t}\langle v_1\ v_2\rangle \boldsymbol{z}$. When this rule is used to generate a critical pair with any other rule, only a variable in $\boldsymbol{t}$ or in $\boldsymbol{z}$ can match, with the whole left-hand side of the other rule. The resulting critical pair s, t reaches confluence (to $s' = t'$) in one step because redexes are non-overlapping. The same holds for [$\mathcal{O}$-reduce].

The only rule pairs candidate for overlapping are [$\mathcal{P}$-reduce] with itself, and [$\mathcal{O}$-reduce] with itself; we only show the first one. The only interesting case of overlapping is the term family $\sum \boldsymbol{t}\langle v_1\ v_2\ v_3\rangle \boldsymbol{z}$, generating the critical pair s, t. Notice that $s' \to t'$ and vice-versa because of the associativity of $\oplus$:

$$\begin{tikzcd}
& \sum \boldsymbol{t}\langle v_1\ v_2\ v_3\rangle \boldsymbol{z} \arrow[dl] \arrow[dr] & \\
s = \sum \boldsymbol{t}\langle (v_1 \oplus v_2) v_3\rangle \boldsymbol{z} \arrow[d] & & \sum \boldsymbol{t}\langle v_1 (v_2 \oplus v_3)\rangle \boldsymbol{z} = t \arrow[d] \\
s' = \sum \boldsymbol{t}\langle (v_1 \oplus v_2) \oplus v_3\rangle \boldsymbol{z} \arrow[rr, shift left] & & \sum \boldsymbol{t}\langle v_1 \oplus (v_2 \oplus v_3)\rangle \boldsymbol{z} = t' \arrow[ll, shift left]
\end{tikzcd}$$ □

Definition 4 (Game tree). *Let $\mathrel{-\!\!\triangleright}$ be the sub-rewrite system of $\to_c$, made only by the rules [$\mathcal{P}$-expand], [$\mathcal{O}$-expand] and [Context]: given an initial position $\pi_0 \in \mathbb{P}$, the set of game tree prefixes from π_0 is the set $T_{\pi_0} = \{t \mid \pi_0 \mathrel{-\!\!\triangleright}^* t\}$. The game tree, if it exists, is the tree $t_{\pi_0} \in T_{\pi_0}$ whose positions are all terminal.*

The game tree t_{π_0} is well-defined: when it exists it is unique. Actually, the TRS defining $\mathrel{-\!\!\triangleright}^*$ is non-ambiguous (there is no *overlap* among any reduction rules) and left-linear: such a TRS is called *orthogonal*, and any orthogonal TRS is confluent [8].

Proposition 2. *$\to_c$ is normalizing for any Nötherian game.*

Proof. Let a game $G = (S, \mathcal{A}, p)$ where $S = (\mathbb{P}, \lambda, succ)$ and $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$ be given. We prove normalization by exhibiting a *reduction order* $<$ compatible with our rules [6].

Let us define a *weight* function $w : \mathbb{P} \to \mathbb{N}$ to be a particular instance of the higher-order function $v_{\bar{p}} : \mathbb{P} \to \mathbb{U}$, where $\bar{p}(\pi) = 2$ for any $\pi \in \mathbb{P}_T$ and $\bigoplus_{i=1}^n x_i = \bigodot_{i=1}^n x_i = 2 + \sum_{i=1}^n x_i$ for any $x \in \mathbb{N}^*$. Intuitively, w returns 2 times the number of nodes in the game tree for Nötherian games.

Let $f : Ter(G) \to \mathbb{N}$ be:

$$\begin{array}{ll} f(\pi) = w(\pi), & \pi \in \mathbb{P} \\ f(v) = 1, & v \in \mathbb{U} \\ f(\Lambda\langle t_1 ... t_n\rangle) = 1 + \sum_{i=1}^n f(t_i) & \end{array}$$

In the formula above and in the rest of this proof $\sum$ represents the sum operation over $\mathbb{N}$. We define our order on terms by using the interpretation f on $>_{\mathbb{N}}$: by definition, let $t_0 > t_1$ iff $f(t_0) >_{\mathbb{N}} f(t_1)$. The order $>$ is trivially *stable*, as our terms do not contain variables. $>$ is also *monotonic* (f is increasing because $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is increasing), *strict* ($>_\mathbb{N}$ is strict) and *well-founded* ($>_\mathbb{N}$ is well-founded). Hence, $>$ is a reduction order.

In order to prove compatibility we show that for every rule $l \to r$ we have $l > r$, which by definition is equivalent to $f(l) >_\mathbb{N} f(r)$. All equalities follow from definitions or trivial algebraic manipulations:

– [Payoff]: $f(\pi) = w(\pi) = \bar{p}(\pi) = 2 >_\mathbb{N} 1 = f(v)$.
– [$\mathcal{P}$-expand], [$\mathcal{O}$-expand]: $f(\pi) = w(\pi) = 2+\sum_{i=1}^{n} w(\pi_i) >_\mathbb{N} 1+\sum_{i=1}^{n} w(\pi_i) = 1 + \sum_{i=1}^{n} f(t_i) = f(\Lambda\ \boldsymbol{t})$.
– [$\mathcal{P}$-reduce], [$\mathcal{O}$-reduce]: $f(\Lambda\, \boldsymbol{t}\langle v_1 v_2\rangle \boldsymbol{z}) = \sum_{i=1}^{\#\boldsymbol{t}} f(t_i)+f(v_1)+f(v_2)+\sum_{i=1}^{\#\boldsymbol{z}} f(z_i) = \sum_{i=1}^{\#\boldsymbol{t}} f(t_i)+1+1+\sum_{i=1}^{\#\boldsymbol{z}} f(z_i) >_\mathbb{N} \sum_{i=1}^{\#\boldsymbol{t}} f(t_i)+1+\sum_{i=1}^{\#\boldsymbol{z}} f(z_i) = f(\Lambda\, \boldsymbol{t}\langle v\rangle \boldsymbol{z})$.
– [Return]: $f(\Lambda\langle v\rangle) = 1 + 1 >_\mathbb{N} 1 = f(v)$. □

Intuitively, if a term converges then its sub-terms also converge; said otherwise if a term converges in a context, then it must also converge in the trivial (empty) context. This is true because of the *non-erasing* nature of our system, different from, for example, the λ-calculus having actual *reduction steps* [8]. More formally:

Lemma 1 (Sub-term normalization). *Given a game $G = (S, \mathcal{A}, p)$ where $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$, for any term $t \in Ter(G)$ and any context C, if there exists $v \in \mathbb{U}$ such that $C[t] \to_c^* v$ then there exists $v' \in \mathbb{U}$ such that $t \to_c^* v'$.*

Proof. By induction over the derivation length n of $C[t] \to_c^* v$. We look at the possible shape of the premise of the [Context] rule, $s \to s'$.

– Base case, $n = 1$: $C[t] \to_c v$. The only applicable rules are [Payoff] and [Return]: in the case of [Payoff], $C[t] = t$; in the case of [Return], $t = v$. In either case, $t \to_c^* v = v'$.
– Recursive case $n \Rightarrow n + 1$: $t_0 = C[t] \to_c t_1 \to_c^* v$. The inductive hypothesis is that for any term t' and context C' if $C'[t'] \to_c^* v$ in n or fewer steps, then $t' \to_c^* v'$. Three cases:
  • s and t are disjoint sub-terms within C. Since the system is non-erasing t has not been erased, i.e. $t_1 = C'[t]$; for inductive hypothesis $t \to_c^* v'$.
  • s contains t. $s \to s'$ may have as its premise [Return], in which case $s = \Lambda\langle v\rangle$ and $t = v$. Otherwise the premise may be [$\mathcal{P}$-Reduce] or [$\mathcal{O}$-Reduce]: either t is one of the values, or it matches one of the variables, in which case there exists a context C' such that $t_1 = C'[t]$; then the inductive hypothesis applies.
  • t contains s. $t = C'[s]$, hence by definition of $\to_c$ we have that t can turn into $C'[s'] = t'$. There exists a context C'' where $C[s] = C''[C'[s]]$, hence $t_1 = C''[C'[s']]$. By induction hypothesis $t' = C'[s'] \to_c^* v'$. □

Normalization and confluence justify our re-definition of the game value v_p as the transitive closure of the transition relation $\to_c$:

Definition 5 (Game Value). *Let a game, an initial position π and a value v be given; we say that the game value from π is v (and we write $v_p(\pi) = v$) if and only if $\pi \to_c^* v$.*

3 α-β pruning

The α-β *algorithm* [1,2] is a method for computing the exact value of a *min-max* combinatorial game without exhaustively visiting *all* game positions.

The α-β algorithm is traditionally presented as a recursive function written in imperative style (see Figure 1): the function `alpha_beta` analyzes a game position $\pi \in \mathbb{P}$ with two additional parameters, α and β, each one denoting a sort of *threshold* not to be overstepped during the incremental computation of the value of $\mathbb{P}$. Whenever the threshold is past the evaluation of an entire subtree is aborted, as it can be proven that it will not contribute to the result.

The correctness of α-β relies on the algebraic properties of the *min* and *max* functions, notably their *mutual* distributive laws — something we can *not* count on under our weaker hypotheses on $\oplus$ and $\odot$ [4,5,3].

```
function alpha_beta($\pi : \mathbb{P};\ \alpha, \beta : \mathbb{Z}$) : $\mathbb{Z}$
  if $\pi \in \mathbb{P}_T$ then
    return $p(\pi)$
  $\pi_1 ... \pi_n := succ(\pi)$  # $n \geq 1$
  if $\lambda(\pi) = \mathcal{P}$ then
    $v := \alpha$
    for $i$ from 1 to $n$
        and while $\beta <_{\mathbb{Z}} v$ do
      $v := min\{v, \text{alpha\_beta}(\pi_i, v, \beta)\}$
  else # $\lambda(\pi) = \mathcal{O}$
    $v := \beta$
    for $i$ from 1 to $n$
        and while $v <_{\mathbb{Z}} \alpha$ do
      $v := max\{v, \text{alpha\_beta}(\pi_i, \alpha, v)\}$
  return $v$
```

```
function tropical($\pi : \mathbb{P};\ \alpha : \mathbb{U}$) : $\mathbb{U}$
  if $\pi \in \mathbb{P}_T$ then
    return $p(\pi)$
  $\pi_1 ... \pi_n := succ(\pi)$  # $n \geq 1$
  if $\lambda(\pi) = \mathcal{P}$ then
    $v := \alpha$
    for $i$ from 1 to $n$ do
      # do not prune at $\mathcal{P}$'s level
      $v := v \oplus \text{tropical}(\pi_i, v)$
  else # $\lambda(\pi) = \mathcal{O}$
    $v := \text{tropical}(\pi_1, \alpha)$  # No $\mathbf{1}_{\mathbb{U}}$
    for $i$ from 2 to $n$
        and while $\alpha \oplus v \neq \alpha$ do
      $v := v \odot \text{tropical}(\pi_i, \alpha)$
  return $v$
```

Fig. 1. Pruning algorithms: traditional α-β pruning vs. tropical α-pruning. Notice that the tropical version has the first iteration of the second loop unrolled, in order not to depend on the existence of a neutral element for $\odot$.

Going back to our game semantics presentation we can model the α-β's behavior by adding four more rules — two per player:

$$[\mathcal{P}\text{-will}]\ \frac{}{\sum\langle\alpha\ [\prod\langle\beta\ (\sum \boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2]\rangle\ \boldsymbol{t}_3\ \rightarrow\ \sum\langle\alpha\ [\prod\langle\beta\ (\sum\langle\alpha\rangle\boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2]\rangle\ \boldsymbol{t}_3}$$

$$[\mathcal{O}\text{-will}]\ \frac{}{\prod\langle\beta\ [\sum\langle\alpha\ (\prod \boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2]\rangle\ \boldsymbol{t}_3\ \rightarrow\ \prod\langle\beta\ [\sum\langle\alpha\ (\prod\langle\beta\rangle\boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2]\rangle\ \boldsymbol{t}_3}$$

$$[\mathcal{P}\text{-cut}]\ \frac{\alpha \oplus \beta = \alpha}{\sum\langle\alpha\ (\prod\langle\beta\rangle\boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2 \rightarrow \sum\langle\alpha\rangle\boldsymbol{t}_2}\qquad [\mathcal{O}\text{-cut}]\ \frac{\beta \odot \alpha = \beta}{\prod\langle\beta\ (\sum\langle\alpha\rangle\boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2 \rightarrow \prod\langle\beta\rangle\boldsymbol{t}_2}$$

The initialization of v at line 6 should be read as a first "virtual" move of the player, whose evaluation is the value α inherited from an ancestor (the grandparent in an alternate-turn game). This explains the rationale of [$\mathcal{P}$-will][2]: whenever subtrees are nested with turns $\mathcal{P}$-$\mathcal{O}$-$\mathcal{P}$, a grandparent may cross two levels and "give" its grandchild its current accumulator as an initialization value. Of course line 10 is the dual version for the opponent and [$\mathcal{O}$-will].

[$\mathcal{P}$-cut] and [$\mathcal{O}$-cut] are a simple reformulation of the *cut conditions* at lines 7 and 11, where the explicit order $<_{\mathbb{Z}}$ disappears[3] from the condition, now expressed as an equality constraint in the rule premise: $\alpha \oplus \beta = \alpha$ represents the fact that the player would prefer α over β. Dually, $\beta \odot \alpha = \beta$ means that the opponent would prefer β over α.

Remark 3 (Non-alternate turn games). Notice that the cut rules can just fire in alternate-turn contexts: this choice simplifies our exposition, but does not limit generality: see Remarks 1 and 2.

The presence of two exactly symmetrical behaviors is quite evident in either presentation; yet what we are interested in showing now is the fact that such duality is quite incidental: it occurs in a natural way in actual two-player games, yet many more search problems lend themselves to be modeled as games despite lacking an intrinsic symmetry.

We can see α-β as the union of two separate techniques applied at the same time, breaking the algebraic symmetry of the player/opponent operations: in the following we are going to eliminate the rules [$\mathcal{O}$-will] and [$\mathcal{O}$-cut], or equivalently to turn `alpha_beta` into `tropical` (see Figure 1), exploiting the weaker properties of *tropical algebras* which only allow *one* threshold α.

4 Tropical games

As we are dealing with a relatively young research topic, it is not surprising that the formalization of tropical algebras has not yet crystallized into a standard form. And since several details differ among the various presentations, we have to provide our own definition:

Definition 6 (Tropical Algebra). *An algebra $(\mathbb{U}, \oplus, \odot)$ is called a tropical algebra if it satisfies the following properties for any a, b and c in $\mathbb{U}$:*

(i) Associativity of $\oplus$: $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
(ii) Associativity of $\odot$: $a \odot (b \odot c) = (a \odot b) \odot c$
(iii) Left-distributivity of $\odot$ with respect to $\oplus$: $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$
(iv) Right-distributivity of $\odot$ with respect to $\oplus$: $(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c)$

[2] "Will" should be interpreted as "bequeath", in the sense of leaving something as inheritance to a descendent.

[3] This is customary with lattices, when an order is derived from a *least-upper-bound* or *greatest-lower-bound* operation.

Some particular choices of $\mathbb{U}$, $\oplus$ and $\odot$ are widely used: the *min-plus algebra* is obtained by defining $\mathbb{U} \triangleq \mathbb{R} \cup \{+\infty\}$, $a \oplus b \triangleq min\{a, b\}$ and, a little counter-intuitively[4], $a \odot b \triangleq a + b$.

Since $\mathbb{U}$ and $\odot$ can also be usefully instantiated in other ways, we will not simply adopt a min-plus algebra; anyway *in practice* we will also choose $\oplus$ to be a minimum on $\mathbb{U}$, which *in practice* will have a total order. This seems to be the only reasonable choice for the applications[5] and helps to understand the idea, yet nothing in the theory depends on the existence of the order. Again, *in practice*, $\oplus$ will return one of its parameters, so if needed we will always be able to trivially define a total order as $x \leq y$ iff $x \oplus y = x$, for any x and y in $\mathbb{U}$. $\oplus$ and $\odot$ will also tend to be commutative *in practice*, making one of the two distributive properties trivial.

We will not make any of the supplementary hypotheses above; on the other hand, we will require the following *rationality hypothesis*[6]:

Definition 7 (Rationality). *Let $(\mathbb{U}, \oplus, \odot)$ be a tropical algebra such that $\mathbf{0} \in \mathbb{U}$ is a neutral element for $\oplus$ and $\mathbf{1} \in \mathbb{U}$ is a neutral element for $\odot$[7]. We call the algebra rational if, for any $x, y, z \in \mathbb{U}$ we have $x \oplus (y \odot x \odot z) = x$.*

Intuitively, the opponent accumulates costs with $\odot$, "worsening" the game value for the player: the player will always choose just x over x "worsened" by something else. Notice that the notion of rationality for two-player games in Game Theory also includes the dual condition $x \odot (y \oplus x \oplus z) = x$; such condition does *not* hold in general for tropical games.

Definition 8 (Tropical Game, Tropical Trees). *A tropical game $G = (S, \mathcal{A}, p)$ is simply a game based on a* rational *tropical algebra $\mathcal{A}$. We call tropical trees all the game trees of a tropical game, and tropical pruning the α-pruning of a tropical tree. A bi-tropical game is a tropical game whose dual $G^{\perp} = (S^{\perp}, \mathcal{A}^{\perp}, p)$ is also tropical, where $\mathcal{A}^{\perp} = (\mathbb{U}, \odot, \oplus)$ if $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$.*

4.1 Soundness of tropical pruning

Proposition 3 (Insertion property). *Let $(\mathbb{U}, \oplus, \odot)$ be a rational tropical algebra. Then for any $x, y, \alpha, \beta \in \mathbb{U}$ we have $\alpha \oplus (\beta \odot x \odot y) = \alpha \oplus (\beta \odot (\alpha \oplus x) \odot y)$.*

Proof (Using associativity implicitly). $\alpha \oplus (\beta \odot (\alpha \oplus x) \odot y) = \{\text{right-distributivity}\}$ $\alpha \oplus (\beta \odot ((\alpha \odot y) \oplus (x \odot y))) = \{\text{left-distributivity}\}$ $(\alpha \oplus (\beta \odot \alpha \odot y) \oplus (\beta \odot x \odot y) = \{\text{rationality}\}$ $\alpha \oplus (\beta \odot x \odot y)$ □

[4] The particular symbols used for indicating $\oplus$ and $\odot$ are justified by the analogy with $+$ and $\cdot$ in how the distributive law works.

[5] Logic programming is an example of an interesting problem lending itself to be interpreted as a combinatorial game on a universe with no total order [3,4]. Anyway the underlying game is a symmetrical *inf-sup* rather than simply tropical.

[6] In lattice theory, the rationality hypothesis is one of the *absorption identities*.

[7] The existence of neutral elements is not strictly necessary, but it simplifies many statements and proofs; without them several results should be given in both "left" and "right" forms.

The insertion property is the semantic counterpart of the rule [$\mathcal{P}$-will]: it explains why we can "transfer" α down in the tree (or more operationally, why we can "start" from the same α when choosing with $\oplus$ two plies below), without affecting the game value.

Definition 9 ($\mathcal{P}$-irrelevance). *Let $(\mathbb{U}, \oplus, \odot)$ be a rational tropical algebra, and let $\alpha, \beta \in \mathbb{U}$. Then we call $x \in \mathbb{U}$ $\mathcal{P}$-irrelevant with respect to α and β if $\alpha \oplus (\beta \odot x) = \alpha$.*

Intuitively, as the value of an opponent-level tree, x can't affect the value of the game because the player will not give the opponent the opportunity to be in that situation: in other word, the current optimal move for the player doesn't change because of x.

Lemma 2 ($\mathcal{P}$-irrelevance). *Let $(\mathbb{U}, \oplus, \odot)$ be a rational tropical algebra, and $\alpha, \beta \in \mathbb{U}$. If $\alpha \oplus \beta = \alpha$ then any $x \in \mathbb{U}$ is $\mathcal{P}$-irrelevant with respect to α and β.*

Proof. $\alpha \oplus (\beta \odot x) = \{\text{hypothesis}\}\ (\alpha \oplus \beta) \oplus (\beta \odot x) = \{\text{associativity}\}\ \alpha \oplus (\beta \oplus (\beta \odot x)) = \{\text{rationality}\}\ \alpha \oplus \beta = \{\text{hypothesis}\}\ \alpha$ □

Definition 10 (Simulation). *Given a tropical game, we say that a term t' simulates a term t, and we write $t \leq t'$, if $t \rightarrow_c^* v \Rightarrow t' \rightarrow_c^* v$.*

Lemma 3 (Tropical $\mathcal{P}$-will simulation). *Given a tropical game $G = (S, \mathcal{A}, p)$ where $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$, for any term sequence $\alpha, \beta \in \mathbb{U}$, $\boldsymbol{t_0}, \boldsymbol{t_1}, \boldsymbol{t_2} \in Ter(G)^*$*

$$\sum\langle \alpha\ [\prod\langle \beta\ (\sum \boldsymbol{t}_0)\rangle\ \boldsymbol{t}_1]\rangle\ \boldsymbol{t}_2\ \leq\ \sum\langle \alpha\ [\prod\langle \beta\ (\sum\langle \alpha\rangle\ \boldsymbol{t}_0)\rangle\ \boldsymbol{t}_1]\rangle\ \boldsymbol{t}_2$$

Proof. By the Sub-term normalization Lemma, if t converges there will exist some value sequences $\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{U}$ such that $\boldsymbol{t}_0 \rightarrow^* \boldsymbol{v}_0$, $\boldsymbol{t}_1 \rightarrow^* \boldsymbol{v}_1$, $\boldsymbol{t}_2 \rightarrow^* \boldsymbol{v}_2$; let us call v_0 the result of $\bigoplus \boldsymbol{v}_0$, v_1 the result of $\bigodot \boldsymbol{v}_1$ and v_2 the result of $\bigoplus \boldsymbol{v}_2$. Then,

$$\begin{tikzcd}
\sum\langle \alpha\ [\prod\langle \beta\ (\sum \boldsymbol{t}_0)\rangle\ \boldsymbol{t}_1]\rangle\ \boldsymbol{t}_2 \arrow[d, "*"'] & & \sum\langle \alpha\ [\prod\langle \beta\ (\sum\langle \alpha\rangle\ \boldsymbol{t}_0)\rangle\ \boldsymbol{t}_1]\rangle\ \boldsymbol{t}_2 \arrow[d, "*"'] \\
\sum\langle \alpha\ [\prod\langle \beta\ v_0\ v_1\rangle]\ v_2\rangle \arrow[d, "*"'] & & \sum\langle \alpha\ [\prod\langle \beta\ (\alpha \oplus v_0)\ v_1\rangle]\ v_2\rangle \arrow[d, "*"'] \\
\alpha \oplus [\beta \odot v_0 \odot v_1] \oplus v_2 \arrow[rr, phantom, "\overset{\{\text{Insertion}\}}{=}"] & & \alpha \oplus [\beta \odot (\alpha \oplus v_0) \odot v_1] \oplus v_2
\end{tikzcd}$$

In the reductions above we implicitly assume that some sequences are non-empty; the proof trivially generalizes to empty $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$ by using neutral elements. □

Lemma 4 (Tropical cut simulation). *Given a tropical game $G = (S, \mathcal{A}, p)$ where $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$, for any term sequence $\alpha, \beta \in \mathbb{U}$, $\boldsymbol{t_0}, \boldsymbol{t_1} \in Ter(G)^*$ we have that if $\alpha \oplus \beta = \alpha$, then $\sum\langle \alpha\ (\prod\langle \beta\rangle\ \boldsymbol{t}_0)\rangle\ \boldsymbol{t}_1\ \leq\ \sum\langle \alpha\rangle\ \boldsymbol{t}_1$.*

Proof. Just like Lemma 3, with $\mathcal{P}$-irrelevance at the end. □

Theorem 1 (Tropical rule soundness). *Adding the rules [$\mathcal{P}$-will] and [$\mathcal{P}$-cut] "does not alter semantics", i.e. if a term t converges to a value v in a system without the two new rules, it is guaranteed to have a reduction sequence converging to v also in the extended system.* □

5 Choose-How-To-Divide and Conquer

According to the classical Divide and Conquer technique a problem can be divided into subproblems, each of which will be solved recursively until a minimal-size instance is found; sub-solutions will then be recomposed.

In the traditional Divide and Conquer style, each division choice is final: it is considered taken once and for all, and cannot be undone. By contrast we present an alternative model based on tropical games. In the *Choose-How-To-Divide and Conquer* style we work with non-deterministic choices in a solution space, using a quality criterion to be optimized and some way of "combining" sub-solutions.

Of course many nondeterministic algorithms can be expressed this way: the challenge is finding a suitable mapping to the tropical game concepts, in term of both syntax and semantics (with the required properties). The problem must have both a suitable *syntactic* structure, and a *semantic* structure with the required properties.

The action of *choosing a division* corresponds to a player node where the $\oplus$ function (typically a minimization) returns the "best" option; the points where *sub-solutions have their cost accumulated* (often something similar to a sum, intuitively "opposed" to $\oplus$) become opponent nodes where $\odot$ combines the values of a subtree sequence into a single result.

Tropical trees have the desirable property of supporting α-pruning, with the potential of significantly cutting down the search space. The more [$\mathcal{P}$-will] and [$\mathcal{P}$-cut] can fire, the more pruning is profitable: hence the problem should be represented as a tropical game having *alternate turns* and *branching factor greater than* 2 *for* $\mathcal{O}$ at least "often enough".
Search problems abound in Artificial Intelligence, and in particular we suspect that more symbolic computation problems than one might expect can be modeled this way. We now proceed to show an unusual application of *Choose-How-To-Divide and Conquer.*

5.1 Parsing as a tropical game

Let $\mathscr{G}$ be a given context-free grammar, defined over an alphabet of terminals $A \ni a$ and nonterminals $N \ni X$. For simplicity[8] let it have no ϵ-production, nor any productions with two consecutive nonterminals or a single nonterminal alone in the right-hand side. Right-hand sides will hence be of the form $[a_1]X_1a_2X_2...a_nX_n[a_{n+1}]$, with $n \geq 0$ and at least one a_i. Given a string of terminals $s \in A^+$ our problem is finding the "best" parse tree of s in $\mathscr{G}$; when s contains some errors our "best" solution just ends up being *the least wrong*, according to some metric; just to keep things simple in this example out metric to minimize will be the total size of the substrings which cannot be matched, in terminals. Sometimes we may wish to have the set of *all* best parses, instead of being content with just one optimal solution.

[8] Such restrictions can be lifted at the cost of some complexity, but supporting a larger class of grammars would be quite inessential for our demonstrative purposes.

Syntax. The set of game positions is defined as $\mathbb{P} = (A^* \times N) \uplus (A^* \times N)^*$, and the turn function is $\lambda(s, X) = \mathcal{P}$, $\lambda((s_1, X_1)...(s_k, X_k)) = \mathcal{O}$. These definitions become easy to understand once the successor function *succ* is examined.

A player position has the form $\pi_\mathcal{P} = (s, X)$, since the player has to parse a string s with a nonterminal X. It has to choose a production $X ::= [a_1]X_1a_2X_2... a_nX_n[a_{n+1}]$, and match the terminals a_i with the terminals in s, in the right order. Each possible match of *all* terminals, for each production of X, is a valid player move generating *strictly smaller* subproblems for the opponent: the nonterminals X_i "in between" the matched terminals will have to be matched to substrings of s in the opponent position $\pi_\mathcal{O} = (s_1, X_1)...(s_1, X_n)$, for some $n \geq 0$. If no match exists with any production then $\pi_\mathcal{P}$ is terminal.

In an opponent position $\pi_\mathcal{O} = (s_1, X_1)...(s_1, X_n)$ the opponent has always exactly n moves: the opponent will give the player each pair (s_i, X_i) to solve "one at the time". For this reason the successor of an opponent position is equal to the position itself: it is the sequence of the elements of $\pi_\mathcal{O}$, itself a sequence. An opponent position $\pi_\mathcal{O}$ is terminal when it is empty.

Figure 2 contains a practical example.

E ::= **n**
E ::= **v**
E ::= **(** E **)**
E ::= **let v** = E **in** E
E ::= **if** E **then** E **else** E
E ::= E = E
E ::= E + E
E ::= E ∗ E

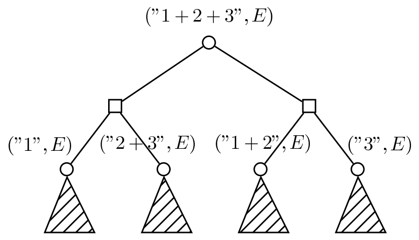

Fig. 2. We use the simple grammar $\mathscr{G}$ above, with an intentionally high level of ambiguity, to parse the string `"1 + 2 + 3"` with E as the start symbol. Circles represent $\sum$ nodes, squares are for $\prod$.

Semantics. We use a *min-plus* algebra for $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$: we simply define $\mathbb{U} \triangleq \mathbb{N}$; we take $\oplus \triangleq min$, since we want as few errors as possible; and finally $\odot \triangleq +$: the number of total errors in the parse tree is equal to the sum of the number of errors in all subtrees.

The payoff $p(\pi)$ is defined as the length in characters of the input string for player positions (notice that the payoff is only defined on *terminal* positions, so such a length is actually the number of unmatched characters), and zero for opponent positions (if $\pi_\mathcal{O} = \langle\rangle$ then there are no errors to accumulate: at the level above, the player matched *the whole* string): $p(s, X) \triangleq \#s$, $p(\langle\rangle) \triangleq 0$.

Experiments We implemented a prototype system[9] in ML supporting the grammar of Figure 2, which can be configured to do a simple exhaustive search or perform tropical α-pruning. The prototype supports two policies: *first-minimal (henceforth FM)* searches for only one optimal strategy at $\mathcal{P}$'s levels, and *all-minimals (henceforth AM)* generates a sequence of strategies with *non-increasing* cost.
Just as illustrative examples, we proceed to show our system behavior on some input strings belonging to different categories.

Non-ambiguous input: the input string `"let x = 42 in x + if 84=42 then 55 else 77"` is parsable in a unique way, so the FM policy is clearly the right choice. Compared to an exhaustive search the α-pruning FM version avoids 98% of the recursive calls (460 vs 28473) and its completion time is 4%. By setting the policy to AM the number of recursive call grows a little, from 460 to 671 (still avoiding 97% of the calls).

Ambiguous input: with the input string `"let x = 84 = 42 = 21 in 1 + 2 * 3"`, which is parsable in several ways, the the α-pruning FM version avoids 99% of the recursive calls (260 vs 61980), and the run time is 1% of the exhaustive-search version time. The α-pruning AM version still avoids 96% of the recursive calls (2148 vs 61980), and its run time is 3%.

"Wrong" input: with the input string `"if if if true then true else false then 10 else (1+(2+)+3)"`, containing errors, the α-pruning FM version avoids 98% of the recursive calls (9640 vs 494344) and its run time is 3%, while the AM version avoids 97% of the recursive calls (13820 vs 494344); the AM version's run time is reduced to 3%. The best strategy has value 6, corresponding to the size of the substring `"if true"` (blanks are not counted) added to the size (0) of the empty substring delimited by the tokens `"+"` and `")"`. The α-pruning algorithm has *localized* errors, guessing that the user should fix her string by replacing `"if true"` with something correct and writing something correct between `"+"` and `")"` — having the size of the unmatched substrings as the payoff function yields this "smallest-incorrect-string" heuristic. Of course other more elaborate criteria are also possible, such as "minimum number of errors".

Memoization: on a completely orthogonal axis, the implementation may be configured to perform *memoization*: when memoization is turned on all the already solved positions are cached, so that they are not computed more than once. We have compared a memoizing version of our tropical-α-pruning parser with a memoizing version performing exhaustive search. In the first case above, the string `"let x = 42 in x + if 84=42 then 55 else 77"` is now parsed with 131 calls instead of 460, again saving 98% of the calls (131 vs 7295) and cutting the run time to 1%. `"let x = 84 = 42 = 21 in 1 + 2 * 3"` is now parsed with 72 calls instead of 260, avoiding 99% of the calls (72 vs 14443) and reducing the run time to 7%. The string `"if if if true then true else false then 10 else (1+(2+)+3)"` is parsed with 1206 calls instead of 9640, avoiding 96% of calls (1206 vs 36575) and cutting the completion time to 10%.

[9] The prototype is freely available under the GNU GPL license at the address `http://www-lipn.univ-paris13.fr/~loddo/aisc-2010`.

At least in our small test cases, tropical α-pruning and memoization work well together: enabling either one does not significantly lessen the efficacy of the other.

6 Conclusions and future work

We have introduced and formally proved correct *tropical α-pruning*, a variant of α-β-pruning applicable to the *tropical games* underlying *Choose-How-To-Divide and Conquer* problems. As a practical example of the technique we have shown how the problem of approximated parsing and error localization can be modeled as a game, and how our pruning technique can dramatically improve its efficiency; yet an asymptotic measure of the visited node reduction would be a worthy development.

We suspect that many more problems can be formalized as tropical games, and the problem of parsing itself can also definitely be attacked in a more general way, lifting our restrictions on the grammar; tropical parsing might prove to be particularly suitable for natural language problems, with their inherent ambiguity.

The correctness and efficiency of *parallel* tropical α-pruning implementations would be particularly interesting to study.

Acknowledgments Christophe Fouqueré first recognized tropical algebras in the properties required by our formalization.

References

1. Hart, T.P., Edwards, D.J.: The tree prune (TP) algorithm. Artificial Intelligence Project Memo 30, Massachusetts Institute of Technology, Cambridge, Massachusetts (1961)
2. Knuth, D.E., Moore, R.W.: An analysis of alpha-beta pruning. Artificial Intelligence **6** (1975) 293–326
3. Loddo, J.V.: Généralisation des Jeux Combinatoires et Applications aux Langages Logiques. PhD thesis, Université Paris VII (2002)
4. Loddo, J.V., Cosmo, R.D.: Playing logic programs with the alpha-beta algorithm. In: Logic for Programming and Automated Reasoning (LPAR). Number 1955 in LNCS, Springer (2000) 207–224
5. Ginsberg, M.L., Jaffray, A.: Alpha-beta pruning under partial orders. In: In Games of No Chance II. (2001)
6. Klop, J.W., de Vrijer, R.: First-order term rewriting systems. In Terese, ed.: Term Rewriting Systems. Cambridge Universisty Press (2003) 24–59
7. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. J. ACM **27**(4) (1980) 797–821
8. Klop, J.W., Oostrom, V.V., de Vrijer, R.: Orthogonality. In Terese, ed.: Term Rewriting Systems. Cambridge Universisty Press (2003) 88–148