Tree-search, Partial Orderings and a New Family of Uninformed Algorithms

Simon Brodt

http://www.pms.ifi.lmu.de/publikationen#PMS-FB-2009-7
Research Report PMS-FB-2009-7
based on the project thesis PA_Simon.Brodt
http://www.pms.ifi.lmu.de/publikationen#PA_Simon.Brodt

Abstract

First a new formalism for the analysis of (uninformed) *search methods* is developed. It connects search methods with *partial orderings*. In this way a characterization of the completeness of a search method and easier completeness checks become possible. Moreover it simplifies the formulation and the proof of further features.

Second a *new uninformed search algorithm* is presented. It is complete and memory efficient and never re-expands nodes. The algorithm is analysed using the previously developed formalism.

Finally advantages of the new method for the *implementation of logic programming languages* are briefly discussed. Especially the efficient processing of (almost) tail recursive progams and the definition of declarative semantics are addressed.

Contents

1. Introduction **5**

2. A Formalism for the Analysis of Traversal-algorithms **5**
- 2.1. Representation of Trees . . . 5
- 2.2. What's an Algorithm? . . . 7
- 2.3. Traversal-orderings . . . 14
- 2.4. Memory . . . 21

3. Known Algorithms in the New Formalism **22**
- 3.1. Depth-First Traversal . . . 22
  - 3.1.1. Representation . . . 22
  - 3.1.2. Incompleteness . . . 22
- 3.2. Breadth-First-Traversal . . . 23
  - 3.2.1. Representation . . . 23
  - 3.2.2. Completeness . . . 24
- 3.3. Iterative Depth-First-Traversal . . . 25
  - 3.3.1. Representation . . . 25
  - 3.3.2. Completeness . . . 25

4. The D&B-traversal and its Family **27**
- 4.1. Intuition . . . 27
- 4.2. The Family . . . 30
- 4.3. Formal . . . 31
  - 4.3.1. Completeness . . . 33
  - 4.3.2. Memory Complexity . . . 37
  - 4.3.3. Memory Complexity of the Family . . . 40

5. Implementation **42**
- 5.1. Approach 1, Depth-bounds . . . 43
- 5.2. Approach 2, Credit-function . . . 45

6. Conclusion **47**

A. Credit-functions **47**

B. Notation **48**

Literatur **48**

1. Introduction

As this is a translation of a German project thesis mainly designed to make the results available to English-speaking readers, the introduction is skipped here.

2. A Formalism for the Analysis of Traversal-algorithms

At the beginning of this chapter probably a certain question arises: Do we really need a new formalism for the analysis of traversal-algorithms? This question is justified as the currently known algorithms may be analysed without such a formalism too. But if you take a closer look at these analyses you will notice that they are very textual. If already these simple algorithms require a significant effort when being analysed without a suitable formalism it is not very promising to try the same for more complex approaches.

We will see that the presented formalism allows fast and brief analyses especially for completeness. This results particularly from the fact that the question of completeness is transferred to an equivalent question on partial orderings, so into a more general and better known field.

But before we are able to start the formalisation and analysis of traversal-algorithms we first have to find a representation for our data, i.e. the trees.

2.1. Representation of Trees

If you want to represent trees with finite maximum branching factor $b \geq 0$ there exists a simple and practical possibility. The idea is to let the name of a node encode just the path from the root to the node. To do so the outgoing edges of each node are numbered $0, \dots, b-1$. The name of a node may be simply constructed by collecting the numbers on the way from the root down to the node. The resulting name is a word out of $\{0, \dots, b-1\}^*$ then.

In the following always consider $\Sigma := \{0, \dots, b-1\}$.

So Σ^* is the set of all possible nodes, and therefore at the same time the node set of the complete infinite tree with branching factor b. We will see this soon.

To be able to define any arbitrary tree the next definition first selects "suitable" node sets $\Omega \subseteq \Sigma^*$.

Definition 2.1.1 (Traversability)**:**

A set $\Omega \subseteq \Sigma^*$ is called *traversable*, if $u \in \Omega$ holds for all $uv \in \Omega$.

The set of all traversable $\Omega \subseteq \Sigma^*$ is denoted by Trav_Σ.

So a node set is "suitable" if for each node also its ancestors are contained in Ω and therefore the node is reachable from the "root" ε.

Finally the definition of a tree:

Definition 2.1.2 (Trees)**:**

Let $E := \{(w, wi) \mid w \in \Sigma^*, i \in \Sigma\}$ then (Σ^*, E) defines a complete infinite tree with branching factor b.

Any tree with maximum out degree b can be obtained by choosing a traversable $\Omega \subseteq \Sigma^*$ and restricting the edge set E to Ω. The resulting tree is:

$$(\Omega, E|_\Omega) = (\Omega, \{(w, w') \in E \mid w, w' \in \Omega\})$$

Instead of $(\Omega, E|_\Omega)$ we will simply write Ω in future.

Remark 2.1.3:

If $\Omega \subseteq \Sigma^*$ is traversable and non-empty, then the following holds:

1. $\varepsilon \in \Omega$ and ε is the root of tree Ω
2. The depth d of a node w in Ω is exactly the length of its name, i.e. $d = |w|$
3. Every prefix w' of a node $w = w'v \in \Omega$ is an ancestor of w. In particular $w' \in \Omega$ is true.

   If $|v| = 1$, then w' is the parent node of w.
4. For each node $w = v_1 v_2 \dots v_k$ in Ω with $k = |w|$ and $v_i \in \Sigma$ holds:

   $\{\varepsilon, v_1, v_1 v_2, \dots, v_1 v_2 \dots v_k\}$ is the path to w in Ω.

   This is exactly the set of prefixes of w and for this w actually encodes the path from the root to itself.

Let's now introduce some abbreviations for the common relations on the nodes of a tree.

Notation 2.1.4:

Consider Ω traversable and $w \in \Omega$. Then define by

- $\mathit{children}_\Omega(w) := \{w' \in \Omega \mid w' = wv \text{ for some } v \in \Sigma\}$

  the set of childrens of w in Ω
- $\mathit{desc}_\Omega(w) := \{w' \in \Omega \mid w' = wv \text{ for some } v \in \Sigma^+\}$

  the set of descendants of w in Ω
- $\mathit{parent}(w) := w'$ with $w'v = w$ for some $v \in \Sigma$

  the parent node of w for $w \neq \varepsilon$
- $\mathit{anc}(w) := \{w' \mid w'v = w \text{ for some } v \in \Sigma^+\}$

  the set of ancestors of w

Frequently one wants to examine a certain level or a certain selection of levels of a tree. The following notation will be useful for this purpose.

Notation 2.1.5:

- $\{\geq n\} := \{i \in \mathbb{N} \mid i \geq n\}$
- $\Omega_k := \{w \in \Omega \mid |w| = k\} = \Omega \cap \Sigma^k$

  $\Omega_N := \{w \in \Omega \mid |w| \in N\} = \bigcup_{i \in N} \Omega_i$ for $N \subseteq \mathbb{N}$

  $\Omega_{\geq k} := \Omega_{\{\geq k\}}$

Since we now have a practical representation of trees, let's turn our attention to algorithms.

2.2. What's an Algorithm?

First an important remark on the notation:

Notation 2.2.1:

1. In the following there is often talked of a ordinal number α with $\alpha \preccurlyeq \omega$. However it should not be necessary to be familiar with ordinal numbers. Such an α simply stands for a set of the form $\{0, 1, \ldots, n\}$ or $\emptyset$ in the finite case ($\alpha \prec \omega$)and $\mathbb{N}$ in the infinite case ($\alpha = \omega$), each together with the common (well-)order on natural numbers. The notation is choosen for two reasons: First it calls attention to the order of the numbers and second the notation avoids to distingiush between finite and infinite case.
2. If Ω und α are used without any furter specification in the future, it is assumed that:
   - $\Omega \subseteq \Sigma^*$ is traversable
   - $\alpha \preccurlyeq \omega$ is an ordinal number

Now let's face traversal algorithms. Surely one characteristic of a traversal-algorithm is the order in which it expands the nodes of a tree.

This will be treated in the upcoming definition. For this some abbreviation will be helpful.

Notation 2.2.2:

Let $a : \alpha \to \Omega$ be a finite or infinite sequence of nodes in Ω.

1. The position of the first occurrence of a node w in a is referred to as $\mathit{first}_a(w)$ and is defined as:

   $\mathit{first}_a(w) := \mathit{min}_<\left(a^{-1}[w]\right) \qquad$ for $w \in a[\alpha]$
2. The position of the first occurrence of a node w in a up from some position $n \in \alpha$ is referred to as $\mathit{first}_a(w, n)$ and is defined as:

   $\mathit{first}_a(w, n) := \mathit{min}_<\left(a^{-1}[w] \cap \{\geq n\}\right) \qquad$ for $w \in a[\alpha \cap \{\geq n\}]$

   (It holds: $\mathit{first}_a(w) = \mathit{first}_a(w, 0)$)

Definition 2.2.3 (Traversal-sequence)**:**

Let $a : \alpha \to \Omega$ be a finite or infinite sequence of nodes in Ω.

a is called *traversal-sequence* if for all w occuring in a also its parent node occurs in a and the first occurence of its parent node is located before the first occurence of w.

Formally:

$$\forall w \in a\left[\alpha\right] \setminus \{\epsilon\} : \underbrace{parent\left(w\right) \in a\left[\alpha\right]}_{parent(w) \text{ occurs in } a} \wedge \mathit{first}_a\left(parent\left(w\right)\right) < \mathit{first}_a\left(w\right)$$

It is shown later, that the above characteristic already determines the completeness of an algorithm. However traversal-sequences don't tell much about another usual concept of algorithms, the one of memory. We can assume, that a traversal-algorithm mainly uses memory for saving a certain set of nodes for later processing. Therefore the memory can be described as a subset of the nodes of a tree.

Definition 2.2.4 (Traversal-runs and traversal-algorithms [1])**:**

- A *traversal-run* $A : \alpha \to \Omega \times \mathcal{P}\left(\Omega\right)$ is a sequence of pairs of a node and a set of nodes for which is true:

  1. The first set of nodes contains exactly the root.
  2. For each pair the node has to be a member of the set of nodes.
  3. In each step the node of the pair is expanded and its children are added to the set of nodes. Arbitrary many nodes previously contained in the set of nodes may be dropped.
  4. If a node w is member of the set of nodes for some certain step, then in the previous step either the set of nodes contained the node itself or its parent node has been expanded.

  Formally:

  For $\alpha = \emptyset$ there is nothing to define.

  Otherwise

  1. $A\left(0\right) = \left(\varepsilon, \{\varepsilon\}\right)$
  2. $\forall n \in \alpha : \left(A\right)_1\left(n\right) \in \left(A\right)_2\left(n\right)$
  3. $\forall n \in \alpha : children_\Omega\left(\left(A\right)_1\left(n\right)\right) \subseteq \left(A\right)_2\left(n+1\right)$
  4. $\forall n \in \alpha \forall w \in \left(A\right)_2\left(n+1\right) : w \in \left(A\right)_2\left(n\right) \vee parent\left(w\right) = \left(A\right)_1\left(n\right)$

  Here $\left(A\right)_i$ is the function, that projects n to the i-th component of $A\left(n\right)$.

- A *traversal-algorithm* is a family $\left(A_\Omega\right)_{\Omega \in Trav_\Sigma}$ of traversal-runs. In other words, the algorithm assigns each tree $\Omega \in Trav_\Sigma$ a traversal-run over this tree.

[1] Commonly one would only talk of an *algorithm*, if the sequence of memory states is computable. But for the following considerations this restriction is unnescesary and potentially troublesome.

Notation 2.2.5:

In the following we often write $A(n)$ instead of $(A)_2(n)$, if it is obvious that $A(n)$ is used in terms of a set.

Each traversal-run can be naturally assigned a traversal-sequence.

Definition 2.2.6 (Induced traversal-sequence)**:**

Let $A : \alpha \to \Omega \times \mathcal{P}(\Omega)$ be a traversal-run. The *induced traversal-sequence* $a_A : \alpha \to \Omega$ of A is defined by:

$$a_A(n) = (A)_1(n)$$

The other way round it would be desirable if we could also obtain a traversal-run for each traversal-sequence. One reasonable possibility (though not the only one) is given by this definition:

Definition 2.2.7 (Induced traversal-run)**:**

Let $a : \alpha \to \Omega$ be a traversal-sequence. The *induced traversal-run* $A_a : \alpha \to \Omega \times \mathcal{P}(\Omega)$ of a is defined by:

$$\begin{aligned} A_a(0) &= (\varepsilon, \{\varepsilon\}) \\ A_a(n+1) &= (a(n+1), (A(n) \backslash D_n) \cup \mathit{children}_\Omega(a(n))) \text{ if } n+1 \in \alpha \end{aligned}$$

where the set D_n is empty or consists of the just expanded node if it (i) will not be expanded again or (ii) will be expanded again only after its parent node will have been expanded another time before.

$$D_n := \begin{cases} \{a(n)\} & \text{if} \quad \underbrace{a(n) \notin a[> n]}_{\text{is never expanded again}} \quad \text{or} \\ & \mathit{parent}(a(n)) \in a[> n] \text{ and} \\ & \mathit{first}_a(a(n), n+1) > \mathit{first}_a(\mathit{parent}(a(n)), n+1) \\ \emptyset & \text{else} \end{cases}$$

Remark 2.2.8:

- If a traversal-run is induced by a traversal-sequence, then its induced traversal-sequence is just the original traversal-sequence.
- A traversal-sequence induced by a traversal-run doesn't necessarily induce the original traversal-run.
- If a traversal-run is induced by a traversal-sequence without repeated expansion then in this traversal-run each node will always be deleted after its (first and unique) expansion. In this way the induced algorithm only saves a minimal amount of nodes at every point of time that still enables completeness.

One of the characteristics of a traversal-algorithm we are most interessted in is its completeness. But what shall completeness mean? The term "completeness"' is mainly used in the context of search algorithms and is there used in two different ways:

An algorithm is called complete if it

1. terminates and returns a solution if there is one (e.g., [Pea84] p.75)
2. returns each solution after a finite amount of time

It's clear that the second possibility implies the first one. In the finite case somehow even the converse is true. If an algorithm is complete in the sense of 1. it can be easily extended to a complete algorithm in the sense of 2. , by restarting the algorithm on the remaining search space after finding a solution.

In the more general context of traversal-algorithms each node is considered to be a solution. So there is no natural way to generalize the first solution. But the generalisation of the second possibility is very simple. For traversal-algorithms it just means that each node (each "solution") is visited after a finite amount of times. [2]

Here the formal definition of completeness:

Definition 2.2.9 (Completeness)**:**

- A traversal-sequence $a : \alpha \to \Omega$ is called *complete*, if it is surjective.
- A traversal-run $A : \alpha \to \Omega \times \mathcal{P}(\Omega)$ is called *complete*, if its induced traversal-sequence is complete.
- A traversal-algorithm is called *complete* on $T \subseteq \mathit{Trav}_\Sigma$, if its runs A_Ω are complete for all $\Omega \in T$.
- A traversal-algorithm is called *finitely complete*, if it is complete on $\{\Omega \in \mathit{Trav}_\Sigma \mid |\Omega| < \infty\}$.
- A traversal-algorithm is called *infinitly complete*, if it is complete on $\{\Omega \in \mathit{Trav}_\Sigma \mid |\Omega| = \infty\}$.
- A traversal-algorithm is called *complete*, if it is as well finitely as infinitely complete. I.e. it is complete on Trav_Σ in the meaning of the third point.

Remark 2.2.10:

- A traversal-run doesn‘t expand any node repeatedly if and only if its induced traversal sequence is injective.
- A complete traversal-run without repeated expansion therefore induces a bijective traversal-sequence.
- If $A : \alpha \to \Omega \times \mathcal{P}(\Omega)$ is a complete traversal-run without repeated expansion, then a node w may not be deleted before its first (and unique) expansion. (results from condition 1 to 4 in Def. 2.2.4).

[2] As a uninformed search algorithm hasn't got the possibility to rule out parts of the search space the 2. definition for search algorithms and the one for traversal algorithms are the same for it. This results from the fact that it has to visit each node to ensure completeness in sense of 2. because it is not able to determine wether an unvisited node is a solution or not.

So the completeness of a traversal-algorithm depends only on its induced traversal-sequence. In the end we would like to use the even more abstract concept of partial orderings for the analysis of the completeness of an algorithm.

When connecting traversal-sequences and orderings the following observation is important:

A traversal-sequence may be considered running through a sequence of points of time and expanding some certain node for each of these points of time. Therefore it is quite natural to order the nodes by the time of their first expansion.

This leads to the term of a *representing ordering*.

Definition 2.2.11:

Let $\lhd$ be a partial ordering of Ω.

1. $\mathit{comp}_{\lhd}(w, w') \;:\Leftrightarrow\; w \lhd w' \vee w = w' \vee w' \lhd w$, tests if w and w' are $\lhd$-comparable with each other
2. $\mathit{Comp}_{\lhd}(w) := \{w' \in \Omega \mid \mathit{comp}_{\lhd}(w, w')\}$, is the set of all nodes $\lhd$-comparable with w.
3. For $M, N \subseteq \Omega$ define:

   $M \lhd N \;:\Leftrightarrow\; \forall m \in M, n \in N : m \lhd n$

Definition 2.2.12 (Representing ordering)**:**

A partial ordering $\lhd$ on Ω is called *representing ordering* of a traversal-sequence a, if the following holds:

- $\forall w, w' \in a[\alpha] : w \lhd w' \Leftrightarrow \mathit{first}_a(w) < \mathit{first}_a(w')$

  All nodes occuring in a are ordered by the point of time of there first occurence
- $a[\alpha] \lhd \mathit{Comp}_{\lhd}(\varepsilon) \setminus a[\alpha]$

If the representing ordering is unique, it is referred to as $\lhd_a$.

Remark 2.2.13:

- A complete traversal-sequence has a unique representing ordering and the second point of the definition is irrelevant. Furthermore the representing ordering is total in this case.
- In the case of an incomplete traversal-sequence there isn't an unique representing ordering. However this ambiguity only results from the different positioning of the elements of $\Omega \setminus a[\alpha]$
- Two different algorithms may have the same ordering or the same orderings.
- If two traversal-sequences without repeated expansion have the same (sets of) representing orderings, then the two traversal-sequences are equal.

In the end we want to use representing orderings for the analysis of completness. Now a condition on traversal-sequences is introduced which on the one hand is necessary for the completeness of the travrsal-ordering and on the other hand guarantees that the representing orderings have some nice properties.

At first glance this might seem a quite strong condition, but we will see immediately after its definition that it can be checked easily in many cases.

Definition 2.2.14 (Weak completeness)**:**

- A traversal-sequence $a : \alpha \to \Omega$ is called *weakly complete*, if $|a\,[\alpha]| = |\Omega|$.
- A traversal-run $A : \alpha \to \Omega \times \mathcal{P}\,(\Omega)$ is called *weakly complete*, if its induced traversal-sequence is weakly complete.
- A traversal-algorithm is called *weakly complete* on $T \subseteq \mathit{Trav}_\Sigma$, if its runs A_Ω are weakly complete for all $\Omega \in T$.
- A traversal-algorithm is called *finitely weakly complete*, if it is weakly complete on $\{\Omega \in \mathit{Trav}_\Sigma \mid |\Omega| < \infty\}$.
- A traversal-algorithm is called *infinitly weakly complete*, if it is weakly complete on $\{\Omega \in \mathit{Trav}_\Sigma \mid |\Omega| = \infty\}$.
- A traversal-algorithm is called *complete*, if it is as well finite as infinite weakly complete. I.e. it is weakly complete on Trav_Σ in the meaning of the third point.

Lemma 2.2.15:

1. Completeness implies weak completeness.
2. Weak completeness implies finite completeness.
3. In the case of uninformed algorithms weak and finite completeness are equivalent.

Proof:

1. : obvious

2. :

follows from the pigeon-hole principle

3. :

additionally to 2. we have to show, that finite completeness implies weak completeness for uninformed algorithms.

The folllowing consideration is important in doing so: After the expansion of finitly many nodes an uninformed algorithm can not determine whether a tree is finite or infinite.

Assume, that the finitly complete algorithm is not weakly complete on an infinite tree Ω. Then obviusly $|a_\Omega\,[\alpha]| < \infty$ holds for the induced traversal-sequence a_Ω. Let $m := \mathit{max}_<\,(\{|w| \mid w \in a_\Omega\,[\alpha]\})$ and so $a_\Omega\,[\alpha] \subseteq \Omega_{\leq m}$. As the algorithm is uninformed, it is not able to distinguish between Ω and $\Omega_{\leq m+1}$, because the algorithm doesn't expand any node out of Ω_{m+1} and above Ω_{m+1}, so in $\Omega_{<m+1}$, the trees are equal. Therefore $a_\Omega[\alpha] = a_{\Omega_{\leq m+1}}[\alpha]$. However $\Omega_{m+1} \neq \emptyset$, as Ω is infinite but finitly branching. On the other hand a_Ω doesn't expand any node in Ω_{m+1} and therefore $a_{\Omega_{\leq m+1}}$ doesn't either. It follows that at least one node in in $\Omega_{\leq m+1}$ isn't visited by $a_{\Omega_{\leq m+1}}$. In other words, $\left|a_{\Omega_{\leq m+1}}[\alpha]\right| < |\Omega_{m+1}| < \infty$. But this contradicts the finite completeness of the algorithm.

$\square$

A natural question is if proving weak completeness is already as hard as proving completeness itself. Fortunatelly it isn't. The following lemma for instance provides a criterion, which allows to prove amongst others the weak completeness of depth- and breadth-first-traversal.

Lemma 2.2.16 (Criterion for weak completeness)**:**

If a traversal-algorithm fulfills the conditions

1. Every node is expanded at most once
2. No node, that hasn't been expanded yet, is deleted
3. The algorithm doesn't stop, while there are still unexpanded nodes left (in memory)

it is weakly complete.

Furthermore proving weak completeness is only a problem when changing from algorithms to orderings. But frequently you do it the other way round. First an ordering is defined, then its completeness is proved and finally you turn to its induced traversal-algorithm. This algorithm is always weakly complete (Remark 2.3.10).

The next section shows what characteristic the representing orderings of a traversal-sequence must have to ensure its completeness.

2.3. Traversal-orderings

It is quite natural to call this characteristic of an ordering *completeness*, too. By means of the term *representing ordering* and the (later defined) *induced traversal-sequence* of an ordering we will finally show in Theorem 2.3.13 the equivalence of the two definitions.

Definition 2.3.1 (Completness of an ordering)**:**

A partial ordering $\lhd$ on $\Omega \subseteq \Sigma^*$ is called *complete*, if and only if $(\Omega, \lhd) \cong \alpha$ for an ordinal number α, $\alpha \preccurlyeq \omega$.

The idea of this definition is that if $(\Omega, \lhd) \cong \alpha$ for an ordinal number $\alpha \preccurlyeq \omega$ there exists no $w \in \Omega$ which is greater (with respect to $\lhd$) than an *infinite* number of other elements of Ω.

The above definition still covers any partial ordering. Though in the context of traversal-algorithms it is suitable to slightly restrict the orderings. Analogously to the restriction on traversal-sequences in Definition 2.2.3 an ancestor of a node should never be greater than the node itself.

Furthermore at least the nodes that are compareable with the root should be ordered nicely, i.e. linearly. This is important for the later definition of the induced traversal-sequence (Definition 2.3.9).

Definition 2.3.2 (Traversal-ordering)**:**

A partial ordering $\lhd$ on traversable $\Omega \subseteq \Sigma^*$ is called *traversal-ordering*, if

- it is compatible with the tree-structure of Ω i.e. for all $uvw \in \Omega$ holds:
  $comp_{\lhd}(u, uvw) \;\Rightarrow\; u \trianglelefteq uv \trianglelefteq uvw$
- $Comp_{\lhd}(\varepsilon)$ is totally ordered by $\lhd$
- $\forall w \in Comp_{\lhd}(\varepsilon), w' \in \Omega \setminus Comp_{\lhd}(\varepsilon) : \neg (w' \lhd w)$

Remark 2.3.3:

- A total traversal-ordering is a topological ordering of Ω
- The representing order $\lhd_a$ of a complete traversal-sequence a is always a traversal-ordering. (Follows immediately from Def. 2.2.3 and Def. 2.2.12)
- If a representing ordering of a (incomplete) traversal-sequence $a : \alpha \to \Omega$ is not a traversal-ordering, then the only cause is the order of the elements of $\Omega \setminus a[\alpha]$.
- For each traversal-sequence there exists a representing traversal-ordering

Up to now we do not benefit from the change from traversal-sequences to traversal-orderings when analysing completeness. The next theorem and the corollary afterwards will change this. But first we need the function defined in Lemma 2.3.4.

Lemma 2.3.4:

If $(\Omega, \lhd)$ is a total ordering and it exists a $f : \mathbb{N} \to \mathbb{N}$ with $\Omega_k \lhd \Omega_{\geq f(k)}$, then the following function is well-defined:
$g : \alpha \to (\Omega, \lhd)$:

$$\begin{aligned} g(0) &:= min_{\lhd}(\Omega) \left(= min_{\lhd}(\Omega \backslash g[\emptyset])\right) \\ g(n+1) &:= min_{\lhd}(\Omega \backslash g[\{0, \ldots, n\}]) \ \text{, if } n+1 \in \alpha \end{aligned}$$

, where $\alpha \preccurlyeq \omega$ and $|\alpha| = |\Omega|$.

Proof:

Let $k := min_{<}(\{|w| \mid w \in \Omega \backslash g[\{0, \ldots, n\}]\})$ and $w \in \Omega \backslash g[\{0, \ldots, n\}]$ with $|w| = k$.
Then $w \lhd \Omega_{\geq f(k)}$.
It follows:

$$\begin{aligned} &\exists w \in \Omega \backslash g[\{0, \ldots, n\}] \, \forall v \in \Omega \backslash g[\{0, \ldots, n\}] : w \unlhd v \\ \Leftrightarrow& \\ &\exists w \in (\Omega \backslash g[\{0, \ldots, n\}]) \cap \Omega_{<f(k)} \forall v \in \Omega \backslash g[\{0, \ldots, n\}] : w \unlhd v \end{aligned}$$

But $(\Omega \backslash g[\{0, \ldots, n\}]) \cap \Omega_{<f(k)}$ is a non-empty set, if $n+1 \in \alpha$ (pigeon-hole principle). Therefore it exists a minimum in it and this minimum is equal to the one of $\Omega \backslash g[\{0, \ldots, n\}]$.

□

Theorem 2.3.5 (Characterisation of Completeness)**:**

A total ordering $(\Omega, \lhd)$ is complete if and only if $\exists f : \mathbb{N} \to \mathbb{N}$ with $\Omega_k \lhd \Omega_{\geq f(k)}$.

Proof:

"$\Rightarrow$":

As $(\Omega, \lhd)$ is complete, there exists an order-preserving isomorphism $h : (\Omega, \lhd) \to \alpha$ with $\alpha \preccurlyeq \omega$.

$$max_k := \begin{cases} 0 & \text{if } \Omega_k = \emptyset \\ max_{<}(h[\Omega_k]) & \text{else} \end{cases}$$

is well-defined, as Ω_k is finite.

Now let $f_k := max_< \left(\{|w| \mid w \in h^{-1}[\{0,\dots,max_k\}]\}\right) + 1$.

Then $\Omega_{\geq f_k} \cap h^{-1}[\{0,\dots,max_k\}] = \emptyset$ and therefore
$min_{f_k} := min_< (h[\Omega_{\geq f_k}]) > max_k$,
because otherwise there exists $w \in \Omega_{\geq f_k}$ with $h(w) \leq max_k$,
so $w \in h^{-1}[\{0,\dots,max_k\}]$ and consequently $f_k > |w|$, a contradiction.

Moreover, as h is an order-preserving isomorphism
$min_\lhd (\Omega_{\geq f_k}) = h^{-1}(min_{f_k}) \rhd h^{-1}(max_k) = max_\lhd (\Omega_k)$ holds and therefore $\Omega_k \lhd \Omega_{\geq f_k}$.

Finally define $f(k) := f_k$.

"$\Leftarrow$"

Choose $\alpha \preccurlyeq \omega$ in such a way, that $|\alpha| = |\Omega|$.

Define $g : \alpha \to (\Omega, \lhd)$ exactly as in Lemma 2.3.4.
It's obvious that g is injective and order-preserving. Especially $(g[\alpha], \lhd) \cong \alpha$.

It's left to show that g is surjective:
(For the finite case this is trivial. In the infinite case we have to ensure that there is no element $w \in \Omega$ which is greater than infinitly many other elements of Ω because the g constructed above never would reach such an w.)

To do so let $w \in \Omega$, $|w| =: k$. Then:

$$\begin{aligned} & w \in \Omega_k \lhd \Omega_{\geq f(k)} \\ \Rightarrow & \{v \in \Omega \mid v \unlhd w\} \subseteq \Omega_{<f(k)} \subseteq \Sigma^{<f(k)} \\ \Rightarrow & \{v \in \Omega \mid v \unlhd w\} \text{ is a finite set} \end{aligned}$$

Let $m := |\{v \in \Omega \mid v \unlhd w\}| \geq 1$.

Show by induction on m:

$$\begin{aligned} & g(m-1) = w \\ & \text{and} \\ & g[\{0,\dots,m-1\}] = \{v \in \Omega \mid v \unlhd w\} \end{aligned}$$

Initial step $m = 1$:

$$\begin{aligned} & m = 1 \\ \Rightarrow & \{v \in \Omega \mid v \unlhd w\} = \{w\} \\ \Rightarrow & w = min_\lhd (\Omega) \\ \Rightarrow & w = g(0) \text{ und } g[\{0\}] = \{w\} = \{v \in \Omega \mid v \unlhd w\} \end{aligned}$$

Induction step $m + 1$:

Let $w' := max_{\lhd}(\{v \in \Omega \mid v \lhd w\})$. Then

$$\begin{aligned} & |\{v \in \Omega \mid v \unlhd w'\}| \\ = & \ |\{v \in \Omega \mid v \lhd w\}| \\ = & \ |\{v \in \Omega \mid v \unlhd w\} \setminus \{w\}| \\ = & \ m \end{aligned}$$

$$\begin{aligned} \text{Induction hypothesis } & \Rightarrow g[\{0,\dots,m-1\}] = \{v \in \Omega \mid v \unlhd w'\} = \{v \in \Omega \mid v \lhd w\} \\ & \Rightarrow g(m) = min_{\lhd}(\Omega \backslash g[\{0,\dots,m-1\}]) = w \\ & \Rightarrow g[\{0,\dots,m\}] = g[\{0,\dots,m-1\}] \cup \{g(m)\} = \{v \in \Omega \mid v \unlhd w\} \end{aligned}$$

□

Corollary 2.3.6:

If $(\Omega, \lhd)$ is a traversal-ordering, then for the right side of the equivalence the following holds additionally:

1. It suffices that $\Omega_k \lhd \Omega_{f(k)}$.
2. w.l.o.g. f is monotonically nondecreasing

Proof:

1.

Show: If $(\Omega, \lhd)$ is a traversal-ordering, then $\Omega_k \lhd \Omega_{f(k)} \Rightarrow \Omega_k \lhd \Omega_{\geq f(k)}$.

For this let $u \in \Omega_{\geq f(k)}$. Then $u = u'v$ for an $u' \in \Omega_{f(k)}$, $v \in \Sigma^*$.
For all $w \in \Omega_k$ the following holds:

$$w \lhd u' \overset{Trav.Ord.}{\unlhd} u'v = u$$

2.

Define $f^*(l) := max_{0 \leq i \leq l}(f(i))$. Then $f^*(l) \geq f(l)$ and $\Omega_{\geq f^*(l)} \subseteq \Omega_{\geq f(l)}$ for all $l \in \mathbb{N}$.

Therefore from $\Omega_l \lhd \Omega_{\geq f(l)}$ follows that $\Omega_l \lhd \Omega_{\geq f^*(l)}$.

Furthermore f^* is monotonically nondecreasing.

□

Put algorithmically the first point of the corollary means, that for every level of the tree there has to exist another deeper level so that non of the nodes at this deeper level is expanded before all nodes of the original level have been expanded.

In the preceding theorem the totality of the examined ordering is required. But if $|\Omega| = \infty$ then the existence of the function f already implies totality.

Corollary 2.3.7:

If $(\Omega, \lhd)$ is a traversal-ordering, $|\Omega| = \infty$ and there exists $f : \mathbb{N} \to \mathbb{N}$ with $\Omega_k \lhd \Omega_{\geq f(k)}$, then $\lhd$ is total and therefore complete.

Proof:

First show by contradiction that $|Comp_{\lhd}(\varepsilon)| = \infty$

Let $m := max_{<}(\{|w| \mid w \in Comp_{\lhd}(\varepsilon)\})$ and $w \in Comp_{\lhd}(\varepsilon)$ with $|w| = m$.

It follows that $\Omega_{\geq f(m)} \cap Comp_{\lhd}(\varepsilon) = \emptyset$.

But as $|\Omega| = \infty$ there exists $w' \in \Omega_{f(m)}$ and because of $w \in \Omega_m \lhd \Omega_{f(m)}$ we have:

$\varepsilon \lhd w \lhd w'$

However this means that $w' \in Comp_{\lhd}(\varepsilon)$, a contradiction to the maximality of m.

Now show by contradiction that $Comp_{\lhd}(\varepsilon) = \Omega$. From this the totality of $\lhd$ on Ω follows immediately as $\lhd$ is total on $Comp_{\lhd}(\varepsilon)$.

For this let $w \in \Omega_k$, $w' \in \Omega_{f(k)} \cap Comp_{\lhd}(\varepsilon)$ (uses $|Comp_{\lhd}(\varepsilon)| = \infty$) for a $k \in \mathbb{N}$. We have $w \lhd w'$.

Then $w \in Comp_{\lhd}(\varepsilon)$ holds, because if not, $\neg(w \lhd w')$ does as $\lhd$ is a traversal-ordering. A contradiction.

□

The following second characterisation of completeness is particularly useful for showing incompletness.

Theorem 2.3.8 (Characterisation of Completeness)**:**

A partial ordering $(\Omega, \lhd)$ is complete if and only if it is isomorphic to a finite sum of complete ordinal numbers, where only the last summand may be infinite (i.e. $= \omega$) [3].

An equivalent proposition is, that $(\Omega, \lhd)$ is complete if and only if $(\Omega, \lhd)$ isomorphic to an countably infinite sum of finite ordinal numbers.

Proof:

The propositions follow immmediatly from the rules of the arithmetic for ordinal numbers. Each of the sums above are equal to an $\alpha \preccurlyeq \omega$.

□

An example for the use of this characterisation can be found in Section 3.1.2 of the next chapter.

After having defined what the completeness of an ordering should mean we want to convince ourself that we have really chosen a suitable definition. So we compare it to the earlier defined completeness of traversal-sequences.

[3] Recall that the addition in the arithmetic for ordinal numbers is generally not commutative.

By the term of representing orderings we are able to change from traversal-sequences to traversal-orderings. The definition of induced traversal-sequences will now establish the connection in the opposite direction.

Definition 2.3.9 (Induced traversal-sequence)**:**

If $(\Omega, \lhd)$ is a traversal-ordering then there is a traversal sequence naturally assigned to it. It is called the *induced traversal-ordering* of $\lhd$ and is defined as follows:

$$\begin{aligned} a_\lhd(0) &:= min_\lhd(Comp_\lhd(\varepsilon)) \\ a_\lhd(n+1) &:= min_\lhd(Comp_\lhd(\varepsilon) \setminus a_\lhd[\{0,\ldots,n\}]) \ \text{, if } n+1 \in \alpha \end{aligned}$$

, where $\alpha \preccurlyeq \omega$ and $|\alpha| = |Comp_\lhd(\varepsilon)|$.

a is well-defined.

Furthermore a is injective (follows immediately from the definition) and actually a traversal-sequence as $\lhd$ is a traversal-ordering.

Proof (Well-definedness)**:**

First of all $Comp_\lhd(\varepsilon)$ is traversable, because $\lhd$ is a traversal-ordering.

Hence $Comp_\lhd(\varepsilon) \setminus a_\lhd[\{0,\ldots,n\}]$ decomposes into finitely many trees $B_1,\ldots,B_l$.

Let $w_1,\ldots,w_l$ be the roots of these trees. As $\lhd$ is a traversal-ordering we know:

$w_i \unlhd B_i$

It follows that $min_\lhd(Comp_\lhd(\varepsilon) \setminus a_\lhd[\{0,\ldots,n\}])$ exists and is equal to $min_\lhd(\{w_1,\ldots,w_l\})$.

□

Remark 2.3.10:

A traversal-sequence induced by a total traversal-ordering is always weakly complete.

The next lemma tackles the problem that a traversal-sequence doesn't necessarily have a unique representing ordering and that not every representing ordering has to be a traversal-ordering (see Remark 2.3.3).

Lemma 2.3.11:

If $\lhd$ and $\lhd'$ are both representing orderings of a traversal-sequence a, then:

1. If one of the representing orderings is incomplete, then a is incomplete, too.
2. If a is weakly complete, then even

   $\lhd$ complete $\Leftrightarrow$ $\lhd'$ complete

   is true.

   Particularly either all representing orderings of a weakly complete traversal-sequence are complete or none.

Remark 2.3.12:

Therefore it suffices to consider a single representing ordering when analysing the completeness of a weakly complete ordering. For this reason we will talk of *the* representing ordering $\lhd_a$ in future even in the case of an incomplete (but weakly complete) traversal-sequence a.

As mentioned earlier weak and finite completeness are equivalent for uninformed traversal-sequences. In other words, uninformed traversal-sequences are already weakly complete if they are finitely complete. So for finitely complete, uninformed traversal-sequences we always have to consider only a single representing ordering $\lhd_a$.

Theorem 2.3.13 (Equivalence of the completeness definitions)**:**

1. A traversal-sequence is complete in the sense of Definition 2.2.9 if and only if all its representing orderings are complete according to Definition 2.3.1.
2. A weakly complete traversal-sequence is complete in the sense of Definition 2.2.9 if and only if its representing ordering is complete according to Definition 2.3.1.

Remark 2.3.14:

- If $\lhd$ is a traversal ordering and $a_\lhd$ the induced traversal-sequence, then $\lhd$ is a representing ordering of $a_\lhd$.
- If $\lhd$ is complete, then the representing ordering of the (complete) induced traversal-sequence $a_\lhd$ is just $\lhd$ itself.

At the end of this section one fact should be emphasized:

To analyse the completeness of an algorithm it suffices to examine its representing orderings. In the majority of cases it even suffices to consider only a single representing ordering.

We will see in the next chapter that by means of the induced algorithm it is often possible to specify a complete algorithm only by its ordering.

2.4. Memory

The definitions of traversal-runs and traversal-algorithms already formalise the concept of memory: A traversal-run assigns each point of time the current set of memorised nodes.

What we still need is a compatible definition of *memory complexity.* The definition will concentrates on the set of memorised nodes and ignores potential administrative information necessary for implementation. Furthermore the definition has to be suitable for infinite trees, even if the algorithm never terminates. The following definition does so. Especially it has the nice property, that it matches the classic definition (maximum of memory consumption over all point of time) in the case of finite trees.

Definition 2.4.1 (Space complexity)**:**

1. The *space complexity* $m_A : \mathbb{N} \to \mathbb{N}$ of a traversal-run A is defined as

   $m_A(d) := \mathit{max}_<(\{|A(n)| \mid n \in \alpha, \mathit{depth}_A(n) \leq d\})$ [4]

   where

   $\mathit{depth}_A(n) := \mathit{max}_<(\{|w| \mid w \in a_A[0, \dots, n]\})$
2. The *space complexity* $M_T : \mathbb{N} \to \mathbb{N}$ *on set* $T \subseteq \mathit{Trav}_\Sigma$ of a traversal algorithm $(A_\Omega)_{\Omega \in T}$ is defined as

   $M_T(d) := \mathit{max}_<(\{m_{A_\Omega}(d) \mid \Omega \in T\})$
3. The *space complexity* $M : \mathbb{N} \to \mathbb{N}$ of a traversal algorithm is defined as

   $M(d) := M_{\mathit{Trav}}(d)$

[4]see notation 2.2.5 (S.9)

3. Known Algorithms in the New Formalism

This chapter makes familiar with the just developed formalism and gives some first feeling of its advantages. For this purpose it is useful to look at the commonly known algorithms in their new "wrapping".

Notation 3.0.2:

Let $(\Omega, \lhd)$ be a partial ordering and $w \in \Omega$.
$\lhd(w) := \{w' \in \Omega \mid w' \lhd w\}$
$\rhd(w) := \{w' \in \Omega \mid w' \rhd w\}$

3.1. Depth-First Traversal

3.1.1. Representation

The depth-first-traversal can be represented easily by the ordering it defines on the nodes of a tree. Using the names we defined for nodes in Section 2.1 the ordering of the depth-first-traversal is just the lexicographical ordering on the names of the node.

In formula:

$\lhd_{depth} = \lhd_{lex}$

or

$\forall w, w' \in \Omega : w \lhd_{depth} w' \Leftrightarrow w \lhd_{lex} w'$

It is not hard to convince oneself that the algorithm induced by this ordering keeps exactly those nodes in memory that a typyical implementation of the depth-first-traversal would keep too.

depth denotes the traversal ordering $\lhd_{depth}$ induced by $a_{\lhd_{depth}}$.[5]

3.1.2. Incompleteness

It is quite obvious that the depth-first-traversal cannot be complete on infinite trees. As a single counter-example suffices to show incompleteness the proof could be done well without the formalism of course. However the following arguments show two things:

[5] This is a very natural notation as $\lhd_{depth}$ is *the* representing ordering of $a_{\lhd_{depth}}$ (or rather *depth*) in sense of remark 2.3.12.

First the formalism allows to put propositions like “The depth-first-traversal never returns from the first infinite branch” precisely.

Second you are able in a sense to “calculate” incompleteness.

But now the formal proof of incompleteness:

If Ω contains an infinite number of nodes it also contains an infinite branch (lemma by König). Let $t \in \Sigma^\omega$ be the lexicographical first infinite branch in Ω. The set of prefixes $T \subseteq \Omega$ of t is just the set of nodes on t. Let $S := \{w \in \Omega \mid T \rhd_{depth} w\}$ and $U := \{w \in \Omega \mid T \lhd_{depth} w\}$. If $U \neq \emptyset$ then the argumentation may be continued in two ways:

Possibility 1 (using Theorem 2.3.5):

$\exists w \in \Omega : T \lhd_{depth} w$

and because of $T \cap \Omega_n \neq \emptyset$ for all $n \in \mathbb{N}$ also

$\exists w \in \Omega \forall n \in \mathbb{N} \exists w' \in \Omega_n : w' \lhd_{depth} w$

It follows ($k = |w|$):

$\exists k \in \mathbb{N} \forall n \in \mathbb{N} : \Omega_k \ntriangleleft_{depth} \Omega_n$

Incompleteness now follows from Theorem 2.3.5.

Possibility 2 (using Theorem 2.3.8):

$S \lhd_{depth} T \lhd_{depth} U$ holds. The smallest ordinal number β that fulfills this condition[6] is $\beta = \underbrace{|S|}_{\cong k \prec \omega} + \underbrace{|T|}_{\cong \omega} + \underbrace{|U|}_{\cong \alpha \succ 0} \succ \omega$

Please note:

In general the lexicographical ordering is not a well-ordering. Particularly $(\Omega, \lhd_{depth}) \cong \beta$ is generally not true. But the consideration above shows, that even if $(\Omega, \lhd_{depth})$ were well-ordered it would still be incomplete.

3.2. Breadth-First-Traversal

3.2.1. Representation

Also the breadth-first-traversal can be easily represented by the ordering it defines. But instead of covering only the breadth-first-traversal the following analysis will cover an abstraction of it, the A^*-Search[7]. This additionally shows that our formalism is useful not only for uninformed but also for informed algorithms.

A characteristic of the A^*-algorithm is its optimistic cost estimation function $f(w) = g(w) + h(w)$. Here g denotes the costs that have as far incurred on the way to w and h the optimistically estimated cost remaining for the way from w to a goal.

[6] An ordinal number β *fullfills* the condition $S \lhd_{depth} T \lhd_{depth} U$, if there is a isomporphic well-ordering $(\Omega, \lhd)$, which fullfills the condition.

[7] The A^*-algorithm doesn't exlude any node from search. It does only prioritize more promissing nodes. When trying to find all solutions it finally traverses the whole tree.

The ordering of the A^*-algorithm is given by:

$w \lhd_{A^*} w' \Leftrightarrow (f(w), w) \lhd (f(w'), w') \Leftrightarrow f(w) < f(w')$ or $f(w) = f(w') \wedge w \lhd_{lex} w'$

Again one can easily convince oneself that the algorithm induced by the ordering above behaves just as expected i.e. like a typical A^*-implementation.

The bradth-first-traversal is obtained by setting $g(w) = |w|$ and $h(w) = 1$.

Then the following holds:

$\Omega_k \lhd_{breadth} \Omega_{k+1}$

$\forall w, w' \in \Omega_k : w \lhd_{breadth} w' \Leftrightarrow w \lhd_{lex} w'$

3.2.2. Completeness

Using theorem 2.3.5 the completeness of the A^*-algorithm can be shown very quickly.

$\Omega_k \lhd_{A^*} \Omega_{(max_<(f[\Omega_k])\,+\,1)}$,

is true because for $w \in \Omega_k$ and $w' \in \Omega_{(max_<(f[\Omega_k])\,+\,1)}$

$f(w') \geq g(w') \geq |w'| = max_<(f[\Omega_k]) \;+\; 1 > max_<(f[\Omega_k]) \geq f(w)$

holds and therefore $w \lhd_{A^*} w'$.

So together with function $k \mapsto max_<(f[\Omega_k]) \;+\; 1$ Theorem 2.3.5 can be applied and the A^*-algorithm is proofed to be complete.

In the special case of the breadt-first-traversal completeness can be shown very quickly by means of arithmetic for ordinal numbers. (Theorem 2.3.8) To do so one only has to convince oneself that

$\lhd_{breadth} \cong \sum_{i=0}^{+\infty} |\Omega_i| \preccurlyeq \omega$

holds.

Compared with the proof in [Pea84] and [Nil80] the argumentation above is extremely short an precise. Due to its formal character it doesn't even need a deeper understanding of the concrete procedure of the A^*-algorithm. So at this point it already shows up that it is useful to analyse algorithms on this more abstract level.

3.3. Iterative Depth-First-Traversal

3.3.1. Representation

Definition 3.3.1:

Let α_1 and α_2 be ordinal numbers, Ω_1 and Ω_2 be traversable and let there exist functions $a_1 : \alpha_1 \rightarrow \Omega_1$ and $a_2 : \alpha_2 \rightarrow \Omega_2$.

The function $(a_1 \,;a_2) : (\alpha_1 + \alpha_2) \rightarrow (\Omega_1 \cup \Omega_2)$ is defined by:

if α_1 is finite

$$(a_1 \,;a_2)\,(n) := \begin{cases} a_1\,(n) & \text{if } n < \alpha_1 \\ a_2\,(n - \alpha_1) & \text{else} \end{cases}$$

or in general

$$(a_1 \,;a_2)\,(n) := \begin{cases} a_1\,(n) & \text{if } n \in I\,[\{1\} \times \alpha_1] \\ a_2\,(n) & \text{if } n \in I\,[\{2\} \times \alpha_2] \end{cases}$$

, where $I : (\{1\} \times \alpha_1 \cup \{2\} \times \alpha_2) \rightarrow \alpha_1 + \alpha_2$ is the isomorphism with $I\,[\{1\} \times \alpha_1] \cong \alpha_1$, $I\,[\{2\} \times \alpha_2] \cong \alpha_2$ and $I\,[\{1\} \times \alpha_1] < I\,[\{2\} \times \alpha_2]$.

Remark 3.3.2:

If a_1 and a_2 are traversal-sequences then $(a_1 \,;a_2)$ is a traversal-sequence too.

$$idepth := depth_{\leq 0}\,; depth_{\leq 1}\,; \cdots ; depth_{max(\{|w| \mid w \in \Omega\})} = \mathop{\Large ;}_{i=0}^{max(\{|w| \mid w \in \Omega\})} depth_{\leq i}$$

Here $depth_{\leq i}$ denotes the traversal-sequence of the depth-first-traversal on tree $\Omega_{\leq i}$.

For $|\Omega| = \infty$ let $max\,(\{|w| \mid w \in \Omega\}) := \infty$.

3.3.2. Completeness

It is easy to see that $\lhd_{breadth}$ is a representing ordering of *idepth*. But before we can restrict the analysis of completeness to $\lhd_{breadth}$ we first have to show the weak completeness of *idepth*. Doing so the following observation is very useful:

Lemma 3.3.3:

If a is a traversal-sequence, b is a weakly complete traversal-sequence and moreover b is a subsequence of a then a is weakly complete too.

As $\lhd_{depth}=\lhd_{lex}$ is total, it follows by Remark 2.3.10 that the traversal-sequence *depth* of the depth-first-search is weakly complete. Furthermore you can easily convince yourself that *depth* is a subsequence of *idepth*. In the finite case for instance $depth = depth_{max(\{|w| \mid w\in\Omega\})}$ holds.

Therefore *idepth* is weakly complete according to Lemma 3.3.3.

So we can now refer to *idepth* (remark 2.3.12) as *the* representing ordering $\lhd_{idepth}=\lhd_{breadth}$.

As known $\lhd_{breadth}$ is complete, consequently $\lhd_{idepth}$ is also complete and finally *idepth* is according to Theorem 2.3.13 point 2.

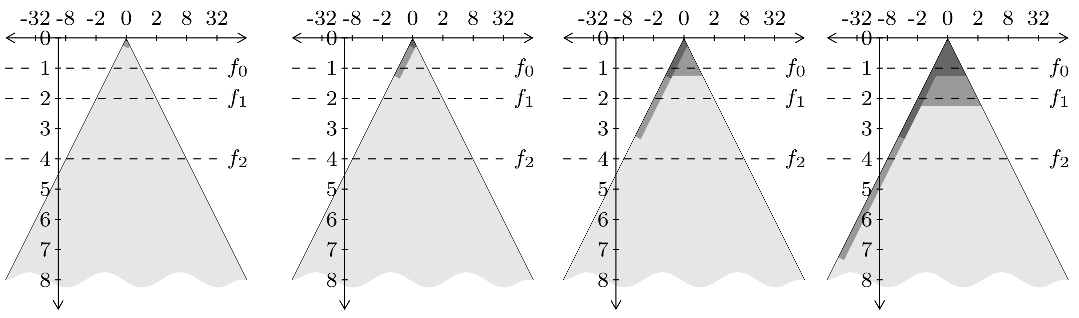
4. The D&B-traversal and its Family

This chapter now presents a possibility to abtain an algorithm that has non of the weaknesses incompleteness, high memory consumption or repeated expansion.

4.1. Intuition

The basic idea behind D&B-traversal is very simple: A depth-first-traversal and a breadth-first-traversal are interleaved. Thereby the depth-first-traversal provides quick advance in the search-space and the breadt-first-traversal guarantees completeness. The actually interesting question is how fast the depth-first-traversal has to and how fast it may advance so that it does not consume too much memory on the one hand but still guarantees completeness on the other hand.

The following sequence of images is to convey a feeling for it.[8]



The images show that the depth-first-traversal advances much faster into depth than the breadth-first-traversal. Though the crucial point is that on the one hand the depth of the depth-first-traversal is bounded by a function of the depth of the breadth-first traversal but on the other hand (just the other way round) the depth of the breadth-first-traversal is bounded by a function of the depth of the depth-first-traversal.[9]Both functions should be monotonously growing and ideally one of them is (essentially) the inverse function of the other. For that reason in following only the first of the two functions mentioned appears, i.e. only the function $i \mapsto f_i$ of the depth of the breadth-first-traversal that bounds the depth of the depth-first-traversal.

Let such a function $i \mapsto f_i$ that fullfills the conditions

$$\begin{aligned} f_i &> i \\ f_{i+1} &> f_i \end{aligned}$$

be given.

[8]Here $f_i := 2^i$.

[9] Plese note that though we speak of depth- and breadth-first-traversal every node is still expanded only once.

Obviously the f_i-th levels Ω_{f_i} then are of great significance. Especially the lexicographically smallest elements of these f_i-th levels play an leading role. They are called *pivot-nodes* because they represent the depth-limits in tree Ω. Based on this idea it is possible to define node sets $S_i \subseteq \Omega$ for each s_i which are traversed one after another as implied by the following pseudo code.

Definition 4.1.1:

$i_{max} := max\left(\{i \mid \Omega_{f_i} \neq \emptyset\}\right)$

for $|\Omega| = \infty$ set $i_{max} := \infty$

$\Omega_{-1} := \emptyset$

$s_i := min_{lex}\left(\Omega_{f_i}\right)$ for $0 \leq i \leq i_{max}$

$$S_i := \left(\underbrace{\lhd_{lex}(s_i)}_{:=D_{i-1}} \cup \underbrace{\Omega_{i-1}}_{:=B_{i-1}}\right) \setminus \underbrace{\bigcup_{j=0}^{i-1} (S_j \cup \{s_j\})}_{:=X_{i-1}} \text{ for } 0 \leq i \leq i_{max}$$

$$R := \Omega \setminus \bigcup_{j=0}^{i_{max}} (S_j \cup \{s_j\})$$

Remark 4.1.2:

- $s_i \in \Omega_{f_i}$ so $|s_i| = f_i$
- Each of the sets S_i respectively R is finite.
- If $i_{max} = \infty$ then $R = \emptyset$.

The D&B-traversal now can be illustrated by the following pseudo code.

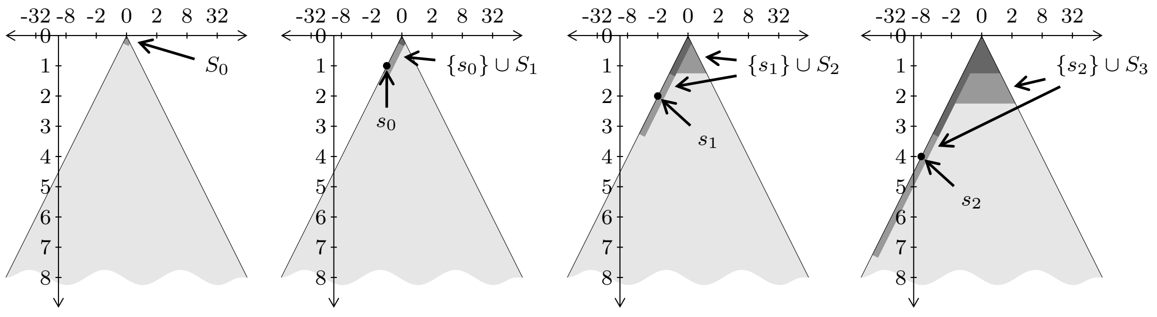
```
while $i < i_{max}$ do
    foreach $w \in S_i$ do [10]
        expand($w$)
    end
    expand($s_i$)
    ++i
end
foreach $w \in R$ do [10]
    expand($w$)
end
```

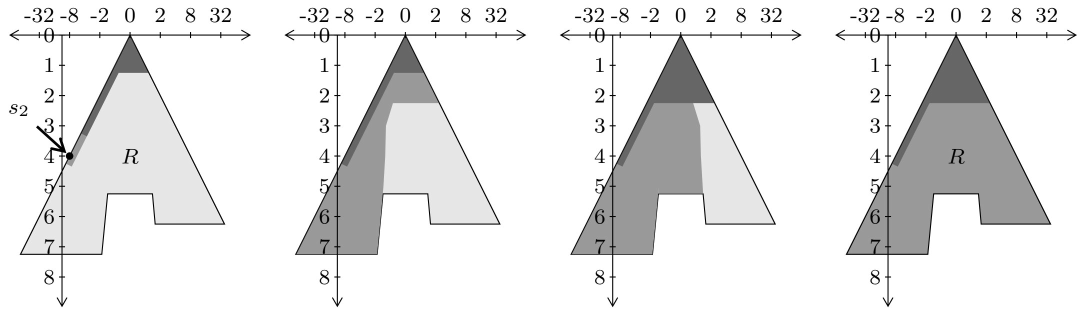
One thing should be added to the code above: It only illustrates the intended effect with regard to the order of expansion. It isn't a possible implementation of the algorithm.

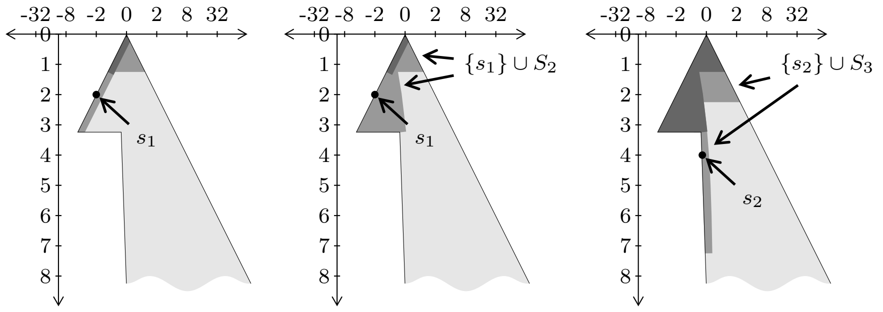
[10] In the end the order in which it is iterated over the nodes is not completely free as the code may suggest. But as the concrete restrictions do not contribute to a better understsnding at this time they are introduced later on page 32.
In the following images it is always iterated over the sets in lexicographical order.

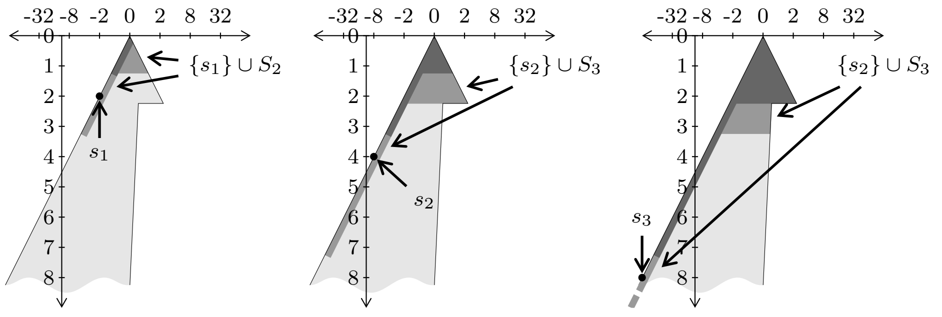
The reason is very simple: An uninformed algorithm only knows the structure of the S_i *after* it already has traversed them. But the code above needs this information *before* the first traversal of the sets. Chapter 5 shows some practicable possibilities for implementation.

The first of the following sequences of images corresponds to the preceding one. The only difference is that the pivot-nodes s_i and the sets S_i are marked instead of the levels f_i. The sequence shows the effect of the definition on the infinite, complete (binary) tree. The three sequences of images after it illustrate the behaviour of the algorithm on other (binary) trees.[11]









[11] The function used in each case is $f_i = 2^i$.
For general trees of degree b analogous images can be obtained with functions $f_i = b^i$

The D&B-traversal has an interesting effect which shows up, when comparing the images for the finite tree with those for the infinite trees. As in a finite tree the depth of the depth-first-traversal is limited by the finite depth of the tree and the depth of the breadth-first-traversal is limited by a function of the depth of the depth-first-traversal the breadth-first-traversal stops after the maximal depth in the tree has been reached. In other words on finite trees the D&B-algorithm does behave almost like the common depth-first-traversal. The other way round on infinite trees the depth-first-traversal "‘gets lost"’ in the first infinite branch and therefore most of the tree is processed by the breadth-first-traversal. One could say that on infinite trees the algorithm behaves almost like a breadth-first-search. If you consider the depth-first-traversal appropriate for finite trees and the breadth-first-traversal appropriate for infinite trees then the D&B-algorithm somehow "‘chooses"’ the appropriate algorithm for the concrete tree. On infinite trees one might prefer to use the iterative depth-first-search instead of the breadt-first-search.

4.2. The Family

Another interesting property of this approach is that using suitable functions f_i a family of algorithms in parameter $c \in \mathbb{N} \cup \{\infty\}$ may be specified which has the following properties:

- For $c \geq 1$ the algorithm is complete.
- For $1 \leq c < \infty$ it has a polynomial space complexity of $O(d^c)$.
- The algorithm for $c = 0$ corresponds to the depth-first-traversal.
- The algorithm for $c = \infty$ corresponds to the breadth-first-traversal.

Practically the required functions can be represented as a single parametrised function $f_{c,i}$ with parameter c. A natural candidate is $f_{c,i} := \lfloor b^{\frac{i}{c}} \rfloor$ [12]. But this function hasn't got all necessary properties (except for $c = 1$), in particular it is not strictly monotonically increasing. However the function $f_{c,i} := \lfloor b^{\frac{i}{c}} \rfloor + i$ [12] is suitable.

Why such parameterability might be useful? By means of parameter c one is able to express how much memory one want to spend for completeness. Now it is possible to choose not only the two extremes, depth- and breadth-first-traversal which spend either all or nothing for completeness. Instead almost arbitrary gradations in-between those two extremes are available. We will see later that the parameter c may be integrated in the implementation. That means, that all algorithms of the whole family only need a single implementation. Actually it is even possible to adapt the parameter dynamically i.e. during the traversal.

Chapter 5 presents such an implementation. The fundamental idea of that implementation derives from the formal analysis of the algorithms which follows now.

[12] Consider $\frac{i}{\infty} = 0$, $\frac{i}{0} = \infty$, $b^{\infty} = \infty$ and $\Omega_{\infty} = \emptyset$.

4.3. Formal

Initially some words on the structure of the formal analysis. At first an axiomatic characterisation of the set based Definition 4.1.1 is given and the equivalence of the original definition and the characterisation is shown. On the one hand the characterisation allows to quickly conclude completeness. On the other hand it is also useful for the proof of the space complexity of the algorithms. Finally it gives us two invariants which will be used in Chapter 5. The proof of space complexity then completes the formal analysis.

At this point please recall the previously introduced Notation 3.0.2 on page 22. It is used extensively in the following.

As the new algorithm is a combination of depth- and breadth-first-traversal, it's a natural idea to reconsider the axiomatic characterisation of those. (see Chapter 3).

The depth-first-traversal is described by the formula

$$\forall w, w' \in \Omega : w \lhd_{depth} w' \Leftrightarrow w \lhd_{lex} w'$$

or alternatively by

$$\forall w \in \Omega_k : \lhd_{lex}(w) \lhd_{depth} w \qquad \text{(DEPTH)}$$

The breadth-first-traversal is characterised by the formulas

1. $\Omega_k \lhd_{breadth} \Omega_{k+1}$
2. $\forall w, w' \in \Omega_k : w \lhd_{breadth} w' \Leftrightarrow w \lhd_{lex} w'$

(BREADTH)

The D&B-traversal can be characterised as follows:

1. $\Omega_k \lhd_{d\&b} \Omega_{f_{k+1}}$
2. $\forall w, w' \in \Omega_k : w \lhd_{d\&b} w' \Leftrightarrow w \lhd_{lex} w'$
3. $\forall w \in \Omega_k : \lhd_{lex}(w) \lhd_{d\&b} w \vee \exists w' \in \Omega_{f_k} : w' \lhd_{d\&b} w$

(D&B) [13]

It's remarkable the formulas above may be interpreted as combination of those of the breadth- and depth-first-traversal. For example the first two formulas are very similar to those of the breadth-first-traversal. The only difference is, that the first formula is a little bit less restrictive (which enables the depth-first-traversal to work). Likewise the first part of the third formula is taken immediatelly from depth-first-traversal. Here the second part of the disjuncton makes the complete formula less restrictive (which enables the breadth-first-traversal to work). By the way the required $w' \in \Omega_{f_k}$ can be considered to be the s_k from Definition 4.1.1.

The functions used above are again restricted as follows:

$$f_i > i$$
$$f_{i+1} > f_i$$

[13] A intuitive meaning of the formulas is given in Chapter 5.

Next Definition 4.1.1 has to be transfered to orderings. Doing so is quite easy. One only needs an ordering $\lhd$ with $S_0 \lhd s_0 \lhd S_1 \lhd s_1 \lhd \cdots \lhd S_{i_{max}} \lhd s_{i_{max}} \lhd R$.

Such an ordering can be obtained by the arithmetic for ordinal numbers as $(\Omega, \lhd) = \sum_{j=0}^{i_{max}} ((S_j, \lhd_j) + \{s_j\}) + (R, \lhd_R)$, where $\lhd_j$ and $\lhd_R$ are linear orderings of the sets S_j and R.

Definition 4.1.1 (S.28) as well as the following pseudo code do not restrict the order in which the nodes of sets S_i and R are traversed. In so far any orderings could be chosen for $\lhd_j$ and $\lhd_R$. Actually these orderings should be restricted a little bit.[14]

The matter is the following: Definition 4.1.1 as well as the images in Section 4.1 show that the sets S_j and R are composed of two parts: First there are the remaining nodes in Ω_{j-1} or $\Omega_{i_{max}}$ i.e. the nodes of the $(j-1)$-th or i_{max}-th level. In each of these levels the nodes should be expanded in lexicographical order ("from left to right"). The part not lying on these levels, so the part of S_j or R reaching into depth, $S_j \backslash \Omega_{j-1}$ or $R \backslash \Omega_{i_{max}}$ should also be expanded in lexicographical order. This is similar to the procedure of the depth-first-traversal.

It should be remarked that this does not determine the order of the expansion of S_j and R strictly. This is only the case on the sets $S_j \cap \Omega_{j-1}$ and $R \cap \Omega_{i_{max}}$ or $S_j \backslash \Omega_{j-1}$ and $R \backslash \Omega_{i_{max}}$. But how the sets $S_j \cap \Omega_{j-1}$ and $S_j \backslash \Omega_{j-1}$ or $R \cap \Omega_{i_{max}}$ und $R \backslash \Omega_{i_{max}}$ are interleaved, is not determined.

Definition 4.3.1 (Addition to Definition 4.1.1)**:**

In the context of Definition 4.1.1 let $\lhd$ be a partial ordering with the following properties:

1. $S_0 \lhd s_0 \lhd S_1 \lhd s_1 \lhd \cdots \lhd S_{i_{max}} \lhd s_{i_{max}} \lhd R$
2. $\forall w, w' \in S_i \cap \Omega_{i-1} : w \lhd w' \Leftrightarrow w \lhd_{lex} w'$
3. $\forall w, w' \in S_i \backslash \Omega_{i-1} : w \lhd w' \Leftrightarrow w \lhd_{lex} w'$
4. $\forall w, w' \in R \cap \Omega_{i_{max}} : w \lhd w' \Leftrightarrow w \lhd_{lex} w'$
5. $\forall w, w' \in R \backslash \Omega_{i_{max}} : w \lhd w' \Leftrightarrow w \lhd_{lex} w'$

Remark 4.3.2:

The requirements of Definition 4.3.1 particularly guarantee that $\lhd$ is a traversal-ordering.

Here an example for an ordering which fullfils the specified conditions. This ordering formalises the strategy of always letting the depth-first-traversal run as long as permited by the current depth of the breadth-first-traversal and then to carry on with the breadth-first-traversal just until it reaches the next greater depth that allows to continue the depth-first-traversal. The images in Section 4.1 used this ordering.

[14] Up to now the restriction of the order in which the nodes of the sets S_i and R are traversed has not been introduced, because it does not contibute to the main idea and would have complicated its representaion.

Example 4.3.3:

$(S_i, \lhd_i) = \left(S_i \cap \lhd_{lex}(s_i), \lhd_{lex}\right) + \left(\left(S_i \setminus \lhd_{lex}(s_i)\right) \cap \Omega_{i-1}, \lhd_{lex}\right)$

$(R, \lhd_R) = (R, \lhd_{lex})$

4.3.1. Completeness

We want to show that Definition 4.1.1 together with Definition 4.3.1 describes exactly the models of (D&B).

First we show that each ordering that satisfies Definition 4.1.1 and 4.3.1 is a model of (D&B).

Lemma 4.3.4:

If $\lhd$ satisfies Definition 4.1.1 and 4.3.1 then

$M := \bigcup\limits_{j=0}^{k} (S_j \cup \{s_j\}) \cup S_{k+1} \ \lhd \ \Omega_{f_{k+1}}$

holds.

Proof:

First we show $M \subseteq \Omega_{<f_{k+1}}$.

Let $w \in M$.

Case $w \in \Omega_i$ for $i \leq k$:

$w \in \Omega_i \subseteq \Omega_{<f_{k+1}}$

Case $w \in \lhd_{lex}(s_i)$ for $i \leq k+1$:

For all $w' \in \Omega_{f_i}$, $s_i \unlhd w'$ holds because $s_i \lhd_{lex} w'$ by definition and so $w' \notin \lhd_{lex}(s_i)$. Furthermore $w' \notin \Omega_j$ for $j \leq i$ as $i < f_i$. Therefore $w' \notin \bigcup\limits_{j=0}^{i-1} (S_j \cup \{s_j\}) \cup S_i$ and thus $w' \ntriangleleft s_i$.

$w \in \lhd_{lex}(s_i) \lhd s_i \unlhd \Omega_{f_i} \overset{\lhd\, Trav.Ordnung}{\unlhd} \Omega_{\geq f_{k+1}}$

Therefore $w \in \Omega_{<f_{k+1}}$.

Case $w = s_i$ for $i \leq k$:

$w = s_i \in \Omega_{f_i} \overset{\text{monotonicity of } f}{\subseteq} \Omega_{<f_{k+1}}$

From this $M \subseteq \Omega_{<f_{k+1}}$ follows, with that also $M \cap \Omega_{f_{k+1}} = \emptyset$ and finally $M \lhd \Omega_{f_{k+1}}$ by the definition of M and Definition 4.3.1(1).

□

Lemma 4.3.5:

If $\lhd$ satisfies Definition 4.1.1 and 4.3.1 then $\Omega_k \lhd \Omega_{f_{k+1}}$ holds.

Proof:

For $k \geq i_{max}$ this is obvious as $\Omega_{f_{k+1}} = \emptyset$.
For $k < i_{max}$ the following holds:
$\Omega_k \subseteq \bigcup\limits_{j=0}^{k} (S_j \cup \{s_j\}) \cup S_{k+1} =:$ M and $M \lhd \Omega_{f_{k+1}}$

$\square$

Lemma 4.3.6:

If $\lhd$ satisfies Definition 4.1.1 and 4.3.1 then $\forall w, w' \in \Omega_k : w \lhd w' \Leftrightarrow w \lhd_{lex} w'$ holds.

Proof:

Case $w, w' \in S_{k+1}$: clear as $w, w' \in S_{k+1} \cap \Omega_k$
Case $w, w' \in S_i$ für $i \leq k$: clear as $w, w' \in S_i \backslash \Omega_{i-1}$
Case $w \in S_i \cup \{s_i\}, w' \in S_j \cup \{s_j\}$ with $i < j \leq k+1$:
Then $w \lhd w'$ as $S_i \cup \{s_i\} \lhd S_j \cup \{s_j\}$. Moreover $w \lhd_{lex} w'$ holds because if $w' \unlhd_{lex} w$ did then $w' \in \unlhd_{lex}(s_i) \unlhd s_i \lhd S_j \ni w'$ follows, a contradiction.
Case $w \in S_i \cup \{s_i\}, w' \in R$ with $i \leq i_{max}$: analogous to the preceding case
Case $w, w' \in R \cap \Omega_{i_{max}}$: clear
Case $w, w' \in R \backslash \Omega_{i_{max}}$: clear

$\square$

Lemma 4.3.7:

If $\lhd$ satisfies Condition 4.1.1 and 4.3.1 then $\forall w \in \Omega_k : \lhd_{lex}(w) \lhd w \vee \exists w' \in \Omega_{f_k} : w' \lhd w$ holds.

Proof:

Case $w = s_i$: clear as $\lhd_{lex}(s_i) \subseteq \bigcup\limits_{j=0}^{i-1} (S_j \cup \{s_j\}) \cup S_i \lhd s_i$

Case $w \in S_i \backslash \Omega_{i-1}$:

$\lhd_{lex}(w) \lhd w$ holds as

$$\lhd_{lex}(w) \backslash S_i = \lhd_{lex}(w) \cap \bigcup_{j=0}^{i-1} (S_j \cup \{s_j\}) \overset{\text{Def. 4.3.1(1)}}{\lhd} w$$

and

$\lhd_{lex}(w) \cap (S_i \backslash \Omega_{i-1}) \lhd w$ because of $\lhd_{lex}(w) \cap (S_i \backslash \Omega_{i-1}) \lhd_{lex} w$ and Definition 4.3.1(3)

and

$\lhd_{lex}(w) \cap (S_i \cap \Omega_{i-1}) \lhd w$ because if $u \in \lhd_{lex}(w) \cap (S_i \cap \Omega_{i-1})$ and $w' \in \Omega_{i-1} \cap anc(w)$ then $u \lhd_{lex} w$ and therefore also $u \lhd_{lex} w'$. From Definition 4.3.1(2) follows that $u \lhd w'$ and as $\lhd$ is a traversal-ordering also $u \lhd w$ is true.

Case $w \in S_i \cap \Omega_{i-1}$ ($\Rightarrow i \geq 1$):

$s_{i-1} \lhd w$ holds. Furthermore $s_{i-1} \in \Omega_{f_{i-1}}$ is true. Therefore the second part of the disjunction is satisfied.

Case $w \in R \cap \Omega_{i_{max}}$: Analogous to the preceding case only with $s_{i_{max}}$ instead of s_{i-1}. Again the second part of the disjunction is satisfied.

Case $w \in R \backslash \Omega_{i_{max}}$:

$$\lhd_{lex}(w) \backslash R = \lhd_{lex}(w) \cap \bigcup_{j=0}^{i_{max}} (S_j \cup \{s_j\}) \lhd w$$

and

$\lhd_{lex}(w) \backslash R \lhd w$ as $\lhd_{lex}(w) \backslash R \lhd_{lex} w$

□

Theorem 4.3.8:

If $\lhd$ satisfies Definition 4.1.1 and 4.3.1 then $\lhd$ is a model of (D&B).

Proof:

Follows immediatelly from the preceding lemmata.

□

In the second part of the formal analysis we show that every model of (D&B) has the structure described in Definition 4.1.1 and 4.3.1.

Theorem 4.3.9:

If $\lhd$ is a model of (D&B) then it satisfies Definition 4.1.1 and 4.3.1.

Proof:

Let $0 \leq i \leq i_{max}$.

1. Show $s_i \lhd \Omega_j$ for $f_i < j$:

   Let $w \in \Omega_j$. Then $w \neq s_i$. Assume $w \lhd s_i$ then

   $w \lhd s_i \in \Omega_{f_i} \lhd \Omega_{f_{f_i+1}}$

   $\underset{f_i+1 \leq j, f_{f_i+1} \leq f_j}{\overset{\lhd\, trav.ordering}{\Rightarrow}} w \lhd \Omega_{f_j}$

   $\Rightarrow \neg\exists w' \in \Omega_{f_j} : w' \lhd w$

   $\overset{\text{(D\&B)(3)}}{\Rightarrow} \lhd_{lex}(w) \lhd w$

   But $s_i \lhd_{lex} w$ because $\unlhd_{lex}(s_i) \subseteq \Omega_{\leq f_i}$ and $w \notin \Omega_{\leq f_i}$

   $\Rightarrow s_i \in \lhd_{lex}(w) \lhd w$ a contradiction.

2. Show $s_i \unlhd \Omega_{\geq f_i}$ and $(w \lhd s_i \Rightarrow w \in \Omega_{<f_i})$:

   From (D&B)(2) follows $s_i \unlhd \Omega_{f_i}$ as $s_i \unlhd_{lex} w$ for all $w \in \Omega_{f_i}$.

   As $\lhd$ is a traversal-ordering we get $s_i \unlhd \Omega_{\geq f_i}$.

   The second proposition follows immediately.

3. Show $\lhd_{lex}(s_i) \lhd s_i$:

   2. and the monotonitcity of f imply

   $\neg\exists w' \in \Omega_{\underbrace{f_{f_i}}_{>f_i}} : w' \lhd s_i$

   und because (D&B)(3) also $\lhd_{lex}(s_i) \lhd s_i$.

4. Show $\lhd_{lex}(w) \lhd w$ for $w \in \Omega_j$ with $w \lhd s_i$ and $i \leq j < f_i$:

   2. and the monotonitcity of f imply

   $\neg\exists w' \in \Omega_{\underbrace{f_j}_{\geq f_i}} : w' \lhd w \quad (w' \lhd w \Rightarrow w' \lhd s_i \Rightarrow w' \in \Omega_{<f_i})$

   $\overset{(D\&B)(3)}{\Rightarrow} \lhd_{lex}(w) \lhd w$

5. Show $s_i \lhd \Omega_j \setminus \lhd_{lex}(s_i)$ for $i \leq j < f_i$:

   Let $w \in \Omega_j \setminus \lhd_{lex}(s_i)$ then $w \notin \lhd_{lex}(s_i)$ and $w \neq s_i$ therefore $s_i \lhd_{lex} w$.
   So if $w \lhd s_i$ did hold, $\lhd_{lex}(w) \lhd w$ wouldn't, a contradiction to 4.

6. Show $\Omega_{i-1} \lhd s_i$:

   $\Omega_{i-1} \overset{\text{(D\&B)(1)}}{\lhd} \Omega_{f_i}$ and $s_i \in \Omega_{f_i}$

7. Show 4.3.1(1), i.e. $S_i \lhd s_i \lhd S_{i+1}$ and $S_{i_{max}} \lhd s_{i_{max}} \lhd R$:

   3. and 6. imply $S_i \subseteq \lhd_{lex}(s_i) \cup \Omega_{i-1} \lhd s_i$.

   Furthermore $s_i \lhd S_{i+1}$ holds because $w \in S_{i+1}$ implies that $w \in \Omega_{\geq i}$ and $w \notin \unlhd_{lex}(s_i)$.
   This means that $w \in \Omega_k \setminus \unlhd_{lex}(s_i)$ for some $k \geq i$ and therefore, according to 5., $s_i \lhd w$.

   The same argument works for $s_{i_{max}} \lhd R$ as $w \in R$ implies that $w \in \Omega_{\geq i_{max}}$ and $w \notin \unlhd_{lex}(s_{i_{max}})$.

   Summarized: $S_i \lhd s_i \lhd S_{i+1}$ and $s_{i_{max}} \lhd R$.

8. Show 4.3.1(2) i.e. $\forall w, w' \in S_i \cap \Omega_{i-1} : w \lhd w' \Leftrightarrow w \lhd_{lex} w'$ and 4.3.1(4) i.e. $\forall w, w' \in R \cap \Omega_{i_{max}} : w \lhd w' \Leftrightarrow w \lhd_{lex} w'$:

   Obvious because of (D&B)(2).

9. Show 4.3.1(3) i.e. $\forall w, w' \in S_i \setminus \Omega_{i-1} : w \lhd w' \Leftrightarrow w \lhd_{lex} w'$:

   $w \in \Omega_{\geq i} \cap \Omega_{<f_i}$ und $w \lhd s_i$ hold for all $w \in S_i \setminus \Omega_{i-1}$. Therefore (as shown in 4.) $\lhd_{lex}(w) \lhd w$ is true.

10. Show 4.3.1(5) i.e. $\forall w, w' \in R \setminus \Omega_{i_{max}} : w \lhd w' \Leftrightarrow w \lhd_{lex} w'$

    For every $w \in R \setminus \Omega_{i_{max}}$ exists some $j > i_{max}$ for which $w \in \Omega_j$
    According to the definition of i_{max}, $\Omega_{f_j} = \emptyset$ holds and therefore $\neg \exists w' \in \Omega_{=f_j} : w' \lhd w$.

    Now (D&B)(3) implies $\lhd_{lex}(w) \lhd w$.

□

Theorem 4.3.10:

If $\lhd$ satisfies Definition 4.1.1 and 4.3.1 or if $\lhd$ is a model of (D&B) then $\lhd$ is complete.

Proof:

Follows immediately from the first formula of (D&B) and Theorems 4.3.8 and 2.3.5.

□

4.3.2. Memory Complexity

Now the space complexity of the presented algorithms is analysed.

For this let $\lhd$ be a model of (D&B), $a_\lhd$ its induced traversal-ordering and $A_\lhd$ its induced traversal-run.

Lemma 4.3.11:

If $a_\lhd(n) \in S_0$ then $|A_\lhd(n+1)| \leq b \cdot (d+1)$,
where $d = max_<(\{|w| \mid w \in a_\lhd[0, \ldots, n]\})$.

Proof:

Because of $\Omega_{i-1} = \emptyset$ and the 3. point of Definition 4.3.1 (p.32) S_0 is traversed in lexicographical order. Therefore $\trianglelefteq_{lex}(a_\lhd(n)) \in a_\lhd[0,\ldots,n]$ holds.

It is an important observation that for all $v \in \lhd_{lex}(a_\lhd(n)) \setminus anc(a_\lhd(n))$, $children(v) \subseteq \lhd_{lex}(a_\lhd(n))$ so also $sons(v) \lhd a_\lhd(n)$ are true. This means, that the children of these nodes have been expanded and aren't still in memory.

So at the most the children of all nodes in $anc(a_\lhd(n)) \cup \{a_\lhd(n)\}$ are in memory, i.e. at the most $|a_\lhd(n)| + 1$ ones.

□

Lemma 4.3.12:

If $a_\lhd(n) \in \{s_i\} \cup S_{i+1}$ then $|A_\lhd(n+1)| \leq b \cdot (d + 1 + b^i)$, where $d = max_<(\{|w| \mid w \in a_\lhd[0,\ldots,n]\})$.

Proof:

$w_{>i,n} := max_{lex}(\Omega_{>i} \cap a_\lhd[0,\ldots,n])$

$w_{i,n} := max_{lex}(\Omega_i \cap a_\lhd[0,\ldots,n])$

1. Show $a_\lhd[0,\ldots,n] = \Omega_{<i} \cup \left(\Omega_i \cap \trianglelefteq_{lex}(w_{i,n})\right) \cup \left(\Omega_{>i} \cap \trianglelefteq_{lex}(w_{>i,n})\right)$:

   $\trianglelefteq_{lex}(s_i) \subseteq \trianglelefteq_{lex}(w_{>i,n}) \subseteq \Omega_{<i} \cup \left(\Omega_i \cap \trianglelefteq_{lex}(u)\right) \cup \left(\Omega_{>i} \cap \trianglelefteq_{lex}(w_{>i,n})\right)$

   ,where $\{u\} = anc(w_{>i,n}) \cap \Omega_i$ and $\Omega_i \cap \trianglelefteq_{lex}(u) \subseteq \Omega_i \cap \trianglelefteq_{lex}(w_{i,n})$.

   Therefore $\bigcup_{j=0}^{i} S_j \cup \{s_j\} \subseteq \Omega_{<i} \cup \left(\Omega_i \cap \trianglelefteq_{lex}(w_{i,n})\right) \cup \left(\Omega_{>i} \cap \trianglelefteq_{lex}(w_{>i,n})\right)$ holds.

   Furthermore $v \trianglelefteq w_{i,n} \Rightarrow v \trianglelefteq_{lex} w_{i,n}$ holds for each $v \in \Omega_i \cap S_{i+1}$. From the maximality of $w_{i,n}$ then follows $\Omega_i \cap a_\lhd[0,\ldots,n] \subseteq \left(\Omega_i \cap \trianglelefteq_{lex}(w_{i,n})\right)$

   Finally $v \trianglelefteq w_{>i,n} \Rightarrow v \trianglelefteq_{lex} w_{>i,n}$ holds for each $v \in \Omega_{>i} \cap S_{i+1}$. From the maximality of $w_{>i,n}$ then follows $\Omega_{>i} \cap a_\lhd[0,\ldots,n] \subseteq \left(\Omega_{>i} \cap \trianglelefteq_{lex}(w_{>i,n})\right)$

   So we have $a_\lhd[0,\ldots,n] \subseteq \Omega_{<i} \cup \left(\Omega_i \cap \trianglelefteq_{lex}(w_{i,n})\right) \cup \left(\Omega_{>i} \cap \trianglelefteq_{lex}(w_{>i,n})\right)$.

   $a_\lhd[0,\ldots,n] \supseteq \Omega_{<i} \cup \left(\Omega_i \cap \trianglelefteq_{lex}(w_{i,n})\right) \cup \left(\Omega_{>i} \cap \trianglelefteq_{lex}(w_{>i,n})\right)$ follows immediately from $w_{>i,n}, w_{i,n} \in a_\lhd[0,\ldots,n]$.

2. Let's now look which nodes the induced traversal-run $A_\lhd$ keeps in memory at some point of time n.

   The first important observation is that $\mathit{children}\left(v\right) \subseteq \lhd_{lex}\left(w_{>i,n}\right)$ holds for all $v \in \lhd_{lex}\left(w_{>i,n}\right) \setminus \mathit{anc}\left(w_{>i,n}\right)$ i.e. that $\mathit{children}\left(v\right) \lhd w_{>i,n}$. The children of these nodes have been expanded already and therefore aren't in memory anymore.

   For every node $v \in \Omega_{<i-1}$ this is obviously true too.

   The children of all nodes $v \in \Omega_{i-1} \cap \lhd_{lex}\left(\mathit{parent}\left(w_{i,n}\right)\right)$ also have been expanded because $\mathit{children}_\Omega\left(v\right) \lhd_{lex} w_{i,n}$.

   All together the children of each node in
   $\mathit{anc}\left(w_{>i,n}\right) \cup \left\{w_{>i,n}\right\} \cup \left(\Omega_i \cap \unlhd_{lex}\left(w_{i,n}\right)\right) \cup \left(\Omega_{i-1} \cap \unrhd_{lex}\left(\mathit{parent}\left(w_{i,n}\right)\right)\right) =: F$ are in memory at the most.

   So $A_\lhd\left(n+1\right) \subseteq \bigcup\limits_{v \in F} \mathit{children}_\Omega\left(v\right)$ holds.

   Now we find

   $\left|\mathit{anc}\left(w_{>i,n}\right) \cup \left\{w_{>i,n}\right\}\right| = \left|w_{>i,n}\right| + 1 \leq d+1$

   and

   $\left|\left(\Omega_i \cap \unlhd_{lex}\left(w_{i,n}\right)\right) \cup \left(\Omega_{i-1} \cap \unrhd_{lex}\left(\mathit{parent}\left(w_{i,n}\right)\right)\right)\right| \leq b^i$.

   This means $\left|A_\lhd\left(n+1\right)\right| \leq b \cdot \left(d+1+b^i\right)$.

□

Lemma 4.3.13:

If $a_\lhd\left(n\right) \in \left\{s_{i_{max}}\right\} \cup R$ then $\left|A_\lhd\left(n+1\right)\right| \leq b \cdot \left(d+1+b^{i_{max}}\right)$,
where $d = \mathit{max}_<\left(\left\{|w| \mid w \in a_\lhd\left[0,\ldots,n\right]\right\}\right)$.

Proof:

Analogously to the preceding lemma only with R instead of S_{i+1}.

□

Theorem 4.3.14:

Let $d \in \mathbb{N}$.

Then the following holds:

$$m_{A_\lhd}\left(d\right) \leq \begin{cases} b \cdot \left(d+1\right) & \text{falls } d < f_0 \\ b \cdot \left(d+1+b^i\right) & \text{sonst} \end{cases}$$

where $i := \arg\max\limits_j \left\{f_j \mid f_j \leq d\right\}$ so i is the j for which f_j is maximal.

Proof:

For $0 \leq i \leq i_{max}$ define $n_i := \mathit{first}_{a_\lhd}(s_i)$.

Let $f_i \leq d < f_{i+1}$ for some $0 \leq i < i_{max}$.

Then $n \leq n_{i+1}$ for $n \in \alpha$, $depth_{A_\lhd}(n) \leq d$ as $depth_{A_\lhd}(n_{i+1}) = f_{i+1} > d$.

Hence $a_\lhd(n) \in \bigcup\limits_{j=0}^{i} (S_j \cup \{s_j\}) \cup S_{i+1}$ and because of Lemma 4.3.12

$m_{A_\lhd}(d) = max_< \left(\left\{|A_\lhd(n)| \mid n \in \alpha, depth_{A_\lhd}(n) \leq d\right\}\right) \leq max_{0 \leq j \leq i}\left(b \cdot \left(d + 1 + b^j\right)\right) = b \cdot \left(d + 1 + b^i\right)$

If $i_{max} < \infty$ and $f_{i_{max}} \leq d$ holds then $a_\lhd(n) \in R$ and therefore analogous to above only with Lemma 4.3.13 instead of Lemma 4.3.12

$m_{A_\lhd}(d) \leq b \cdot \left(d + 1 + b^{i_{max}}\right)$

Furthermore $a_\lhd(n) \in S_0$ holds for $d < f_0$ and so, again analogous to above, with Lemma 4.3.11

$m_{A_\lhd}(d) \leq b \cdot (d + 1)$

□

Corollary 4.3.15:

The space complexity of the algorithm induced by $\lhd$ is

$$M(d) \leq \begin{cases} b \cdot (d+1) & \text{if } d < f_0 \\ b \cdot \left(d + 1 + b^i\right) & \text{else} \end{cases}$$

where $i := \underset{j}{argmax}\{f_j \mid f_j \leq d\}$.

4.3.3. Memory Complexity of the Family

In Section 4.2 on page 30 a family of algorithms in parameter $0 \leq c \leq \infty$ has been presented. For $1 \leq c < \infty$ it has been claimed that the corresponding algorithm had space complexity $O(d^c)$. This will be proved now.

Theorem 4.3.16:

Let $1 \leq c < \infty$, $f_i = f_{c,i} = \lfloor b^{\frac{i}{c}} \rfloor + i$ and $\lhd_c$ be the corresponding ordering. The space complexity of the algorithm induced by $\lhd_c$ is

$M_c(d) \leq b \cdot (d + 1 + d^c)$

Proof:

If $d < f_0$ then $M_c(d) \overset{\text{Cor. 4.3.15}}{\leq} b \cdot (d+1)$.

If $f_0 \leq d < f_1$ then

$M_c(d) \overset{\text{Cor. 4.3.15}}{\leq} b \cdot \left(d+1+b^0\right) = b \cdot (d+1+1) \overset{0<f_0\leq d}{\leq} b \cdot (d+1+d) \leq b \cdot (d+1+d^c)$

If $f_1 \leq d$ then $f_i \leq d$ for $i := \arg\underset{j}{max}\,\{f_j \mid f_j \leq d\}$ with $i \geq 1$.

Therefore $\lfloor b^{\frac{i}{c}} \rfloor + i \leq d$ and because of $i \geq 1$ also $b^{\frac{i}{c}} \leq d$ and by this $b^i \leq d^c$.
From Corollary 4.3.15 follows that
$M_c(d) \leq b \cdot \left(d+1+b^i\right) \leq (d+1+d^c)$

$\square$

As expected for the depth-first-traversal the case $c = 0$ has linear space complexity in the maximum depth reached so far.

Theorem 4.3.17:

Let $c = 0$, $f_i = f_{0,i} = \lfloor b^{\frac{i}{0}} \rfloor + i = \infty$ and $\lhd_0$ be the corresponding ordering. The space complexity of the algorithm induced by $\lhd_0$ is
$M_0(d) \leq b \cdot (d+1)$

Proof:

$f_0 = \infty$ holds, so $d < f_0$ for all $d \in \mathbb{N}$ and therefore
$M_0(d) \overset{\text{Cor. 4.3.15}}{\leq} b \cdot (d+1)$.

$\square$

5. Implementation

Up to this point we have seen the idea of the new algorithm and have analysed its properties. But there is still to show that the algorithm can be implemented. Two possibilities are presented in this chapter.

Approach 1 closely follows the formulas of (D&B) on page 31. Therefore it is obvious that they are valid. However approach 1 needs to know the maximum branching factor b.

Approach 2 does not need this information. However it does not obey stricly to the formulas of (D&B). But one can convince oneself, that the variations do neither affect completeness nor the space complexity of the algorithm.

Both implementations will use a memory structure Ω in their pseudo code whose content is adressed in form $\Omega[i]$ by an index i. The name is chosen carefully as the nodes referenced by $\Omega[i]$ are just the nodes of Ω_i that are in memory currently. In other words $\Omega[i] = \Omega_i \cap A_{d\&b}(n)$ for each point of time n. This memory structure may be implemented by a list of lists (both doubly linked).

The lexicograhical ordering in all of these lists $\Omega[i]$ is simply guaranteed by the fact that the nodes come up just in this order and are added to the end of the list. This holds because the method for the expansion of a node is only called in the form 'expand(**first of** Ω[i])' and so always the lexicographical smallest element of Ω[i] is expanded. For this reason '**first of** Ω[i]' does not only denote the lexicographical smallest element of Ω[i] but also just the head of the list. Moreover the property above, that only the lexicographical smallest element of a list is expanded, implies that the second formula of (D&B) is always satisfied. After all the formula just means that each level is traversed in lexicographical order. Furthermore this guarantees that the first part of the disjunction of the third formula of (D&B) is satisfied for each node expanded by depth-first-traversal. This part says that a node may only be expanded if all nodes left of it in the tree have been expanded before.

Here the pseudo code of the method for the expamsion of a node. It is used by both implementations.

```
expand(w)
    Ω[$|w|$] := Ω[$|w|$] \ $\{w\}$
    Ω[$|w|$ + 1] := Ω[$|w|$ + 1] ∪ children(w)
    unexpandedNodes := unexpandedNodes − 1 + $|children(w)|$
end
```

5.1. Approach 1, Depth-bounds

As announced at the beginning of this chapter the following implementation is closely adapted to the formulas of (D&B). So it is useful to point out the intuitive meaning of the formulas of (D&B) first.

The first axiom $\Omega_k \lhd_{d\&b} \Omega_{f_{k+1}}$ says that none of the nodes in $\Omega_{f_{k+1}}$ is expanded before all nodes of Ω_k have been expanded. Therefore it limits the depth of the depth-first-traversal.

The third axiom is a quantified disjunction. The second part of the disjunction concerns the breadth-first-traversal. It means that a node may only be expanded if some node at sufficient depth has been expanded (by depth-first-traversal) before. As one of the two parts of the disjunction has to be true for every node this is the same as a depth bound for breadth-first-traversal.

This interpretaion of the formulas as mutual depth-bounds is used by the implementation. You can clearly identify the depth-bound in the code by b_{depth} and $b_{breadth}$. They are increased (forth case of the while-loop) if a so-called “pivot-node” is expanded. These pivot-nodes are the s_i of Definition 4.1.1. They are obvious in the images in Section 4.1.

In the first three cases of the while-loop it is obvious that the depth-bounds are respected. In the forth case we have the following: As $d_{breadth} = b_{breadth}$ the breadth-first-traversal has just finished the one level and $d_{breadth}$ has been incremented by 1. The strict monotonicity of f implies $\mathrm{f}(d_{breadth}) > b_{depth}$. So the depth-first-traversal may advance into depth at least one step. Therefore the depth-bounds are respected when calling depthStep().

Though the following implemtation has a disadvantage. The function f used by it depends on the maximum branching factor b of the tree. If b is unknown this approach is not directly applyable. It can be repaired by always using for f the maxium branching factor that occured so far. Nevertheless a implementation that is independent of the branching factor would be desirable. Such an implementation is presented in the next section.

```
$\Omega[0] := \{\varepsilon\}$
unexpandedNodes := 1

$d_{depth}$   := 0      (*current depth of depth-first-traversal*)
$d_{breadth}$ := 0      (*current depth of breadth-first-traversal*)
$b_{depth}$   := f(0)   (*current depth bound of depth-first-traversal*)
$b_{breadth}$ := 0      (*current depth bound of breadth-first-traversal*)

depth_breadth_traversal()
    while $unexpandedNodes > 0$ do
        if $d_{depth} < b_{depth}$ and $d_{breadth} < b_{breadth}$ then do
            one of depthStep() or breadthStep()
        else if $d_{depth} < b_{depth}$ then do
            depthStep()
        else if $d_{breadth} < b_{breadth}$ then do
            breadthStep()
        else do (*$d_{depth} = b_{depth}$ and $d_{breadth} = b_{breadth}$*)
            depthStep()
            if $d_{depth} > b_{depth}$ do (*pivot-node has been expanded*)
                ++ $b_{breadth}$
                $b_{depth}$ := f($d_{breadth}$)
            end
        end
    end
end

depthStep()
    if $\Omega[d_{depth}] \neq \emptyset$ then do
        expand(first of $\Omega[d_{depth}]$)
        ++ $d_{depth}$
    else do
        -- $d_{depth}$ (*backtrack*)
    end
end

breadthStep()
    if $\Omega[d_{breadth}] \neq \emptyset$ then do
        expand(first of $\Omega[d_{breadth}]$)
        $d_{depth}$ := max($d_{depth}$, $d_{breadth}$ + 1)
    else do
        ++ $d_{breadth}$
    end
end
```

5.2. Approach 2, Credit-function

The goal of the following implemetation is to reproduce the behavior of the preceeding implemetation as closely as possible but without using the maximum branching factor b.

To reach this goal two observations are important:

First in the algorithm described by (D&B) not only the number of nodes in memory is bounded polynomially by the maximum depth reached so far but also the number of nodes expanded by breadth-first search until then (More precicely, the second bound is at most the same size as the first and this is independet of b) .

Second a polynomial bound of the number of nodes expanded by breadth-first-traversal immediately implies a polynomial bound of the number of nodes in memory.

So its an obvious idea, to realize the polynomial bound of memory consumption in the maximum depth reached so far not by limiting the *depth* of the breadth-first-traversal but the *number of nodes* expanded by it.

Also the depth bound for the depth of the depth-first traversal can be replaced. If one forces the number of nodes expanded by breadth-first-search be bounded by the maximum depth reached so far not only upwards but also downwards then this results in a suitable limitation of the depth of the depth-first-traversal.

In the following pseudo code $c_{breadth}$ is the number of additional nodes that the breadth-first-traversal may expand currently. $c_{breadth}$ is increased each time depth-first-traversal reaches a new maximum depth. If $c_{breadth}$ is big enough that the breadth-first-traversal may finish its current level, i.e. if $c_{breadth} \geq |\Omega\,[d_{breadth}]|$, then the depth-first-traversal is paused, which limits the depth of the depth-first-traversal.

For this $\mathrm{credit}\,(\mathrm{d})$ needs to satisfy $\sum\limits_{\mathrm{i}=0}^{\mathrm{d}} \mathrm{credit}\,(\mathrm{i}) = O(\mathrm{d}^c)$.

This can be achieved most easily by setting $\mathrm{credit}\,(0) = 0$ and $\mathrm{credit}\,(\mathrm{d}) = \mathrm{d}^c - (\mathrm{d}-1)^c$ for $\mathrm{d} > 0$. However this possibility is hard to be implemented efficiently.

The definition $\mathrm{credit}\,(\mathrm{d}) = (2^c - 1) \cdot 2^{(c-1)\lfloor \log_2(\mathrm{d}) \rfloor}$ may be implemented efficiently by use of shift operations and does also satisfiy the requirements (See Appendix A).

On complete trees this implementation behaves quite exactly as the preceeding implementation. On less dense tress however the breadth-first-traversal advances faster. Beside the fact that the maximum branching factor of the tree isn't needed this is another advantage of the implementation presented here.

```
$\Omega[0] := \{\varepsilon\}$
unexpandedNodes := 1

$d_{depth}$       := 0    (* current depth of depth-first-traversal *)
$d_{breadth}$     := 0    (* current depth of breadth-first-traversal *)
max_$d_{depth}$   := 0    (* maximum depth reached *)
$c_{breadth}$     := 0    (* credit of breadth-first-traversal *)

depth_breadth_traversal()
    while $unexpandedNodes > 0$ do
        if $c_{breadth} < |\Omega[d_{breadth}]|$ and $c_{breadth} > 0$ then do
            one of depthStep() or breadthStep()
        else if $c_{breadth} < |\Omega[d_{breadth}]|$ then do
            depthStep()
        else if $c_{breadth} > 0$ then
            breadthStep()
        else do
            (* impossible case *)
        end
    end
end

depthStep()
    if $\Omega[d_{depth}] \neq \emptyset$ then do
        expand(first of $\Omega[d_{depth}]$)
        if $d_{depth}$ = max_$d_{depth}$ then do
            ++ max_$d_{depth}$
            $c_{breadth} := c_{breadth}$ + credit(max_$d_{depth}$)
        end
        ++ $d_{depth}$
    else do
        -- $d_{depth}$ (* backtrack *)
    end
end

breadthStep()
    if $\Omega[d_{breadth}] \neq \emptyset$ then do
        expand(first of $\Omega[d_{breadth}]$)
        -- $c_{breadth}$
        $d_{depth} := \max(d_{depth}, d_{breadth} + 1)$
    else do
        ++ $d_{breadth}$
        if($d_{breadth}$ = max_$d_{depth}$) then do
            $c_{breadth} := 0$
        end
    end
end
```

6. Conclusion

Again as this is a translation of a German project thesis mainly designed to make the results available to English-speaking readers this part is dropped.

A. Credit-functions

Theorem A.0.1:

Let g and h be monotonic functions from $\mathbb{N}$ to $\mathbb{N}$ and $k_1 \leq k_2 \leq k_3 \leq \ldots$ an increasing sequence in $\mathbb{N}$.

If additionally $g(k_i) \leq h(k_i)$ for all $i \in \mathbb{N}$ and exists $c \in \mathbb{R}$ so that $h(k_{i+1}) \leq c \cdot h(k_i)$ holds for all $i \in \mathbb{N}$, then:

$g = O(h)$

more precisely:

$g \leq c' \cdot h$ for some $c' \leq c$

Proof:

Show: $g(n) \leq c \cdot h(n)$ for all $n \in \mathbb{N}$

Let $k_i \leq n < k_{i+1}$. Then:

$$g(n) \overset{\substack{\text{monotonicity}\\\text{of } g}}{\leq} g(k_{i+1}) \overset{Ass.}{\leq} h(k_{i+1}) \overset{Ass.}{\leq} c \cdot h(k_i) \overset{\substack{\text{monotonicity}\\\text{of } h}}{\leq} c \cdot h(n)$$

□

In the following consider $\mathrm{credit}(0) = 0$ and $\mathrm{credit}(\mathrm{d}) = \mathrm{d}^c - (\mathrm{d}-1)^c$ for $\mathrm{d} > 0$.

Lemma A.0.2:

$$\sum_{\mathrm{i}=0}^{2^{l+1}} \mathrm{credit}(\mathrm{i}) < \left(1 + 2^{c-(l+1)}\right) \cdot \left(2^{l+1}\right)^c$$

Proof:

$$\begin{aligned}
\sum_{i=1}^{2^{l+1}-1} (2^c - 1) \cdot 2^{(c-1)\lfloor \log_2(i) \rfloor} &= \sum_{i=0}^{l} \sum_{j=1}^{2^i} (2^c - 1) \cdot 2^{(c-1)\cdot i} \\
= \sum_{i=0}^{l} 2^i \cdot (2^c - 1) \cdot 2^{(c-1)\cdot i} &= (2^c - 1) \cdot \sum_{i=0}^{l} (2^c)^i \\
= (2^c - 1) \cdot \frac{(2^c)^{l+1} - 1}{2^c - 1} &= \left(2^{l+1}\right)^c - 1
\end{aligned}$$

$$\sum_{i=1}^{2^{l+1}} (2^c-1)\cdot 2^{(c-1)\lfloor \log_2(i)\rfloor} = \left(2^{l+1}\right)^c - 1 + (2^c-1)\cdot 2^{(c-1)\cdot(l+1)}$$
$$< \left(2^{l+1}\right)^c - 1 + 2^c \cdot \left(2^{l+1}\right)^{(c-1)} = \left(1+2^{c-(l+1)}\right)\cdot\left(2^{l+1}\right)^c - 1$$

□

Theorem A.0.3:

$\sum\limits_{\mathrm{i=0}}^{\mathrm{d}} \mathrm{credit}\,(\mathrm{i}) = O(d^c)$

Proof:

Let $g\,(d) := \sum\limits_{\mathrm{i=0}}^{\mathrm{d}} \mathrm{credit}\,(\mathrm{i})$ and $h\,(d) := (1+2^{c-\lfloor \log_2(d)\rfloor})d^c$. Furthermore let $k_i := 2^{i+1}$.

Then $g\,(k_i) \le h\,(k_i)$ holds according to Lemma A.0.2.

Moreover $h\,(k_{i+1}) = \left(1+2^{c-(i+2)}\right)\cdot\left(2^{i+2}\right)^c \le \left(1+2^{c-(i+1)}\right)\cdot\left(2\cdot 2^{i+1}\right)^c = 2^c \cdot h\,(k_i)$

From Theorem A.0.1 follows $g\,(d) = O(h) = O(d^c)$

□

B. Notation

$\Sigma^0 = \{\varepsilon\}$

$\Sigma^1 = \Sigma$

$\Sigma^n = \{uv \mid u \in \Sigma^{n-1},\ v \in \Sigma\}$

$\Sigma^* = \bigcup\limits_{n\in\mathbb{N}} \Sigma^n$

$\Sigma^+ = \bigcup\limits_{n\ge 1} \Sigma^n = \Sigma^* \setminus \{\varepsilon\}$

References

[Nil80] N. J. Nilsson. *Principles of artificial intelligence.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1980.

[Pea84] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.