Search for More Declarativity
Backward Reasoning for Rule Languages Reconsidered

Simon Brodt  François Bry  Norbert Eisinger

Institute for Informatics, University of Munich,
Oettingenstraße 67, D-80538 München, Germany
http://www.pms.ifi.lmu.de/

25 October 2009
Declarativity – The Greatest Advantage of Rule Languages

- Separates between
  - *What* is the problem?
  - *How* is the problem solved?

- Built-in problem-solving
  ⇒ Allows to concentrate on problem-specification

- Add and modify rules easily

- Supports rapid prototyping and stepwise refinement

- Finding solutions where no explicit algorithm is known

- Adaption to frequently changing prerequisites
Declarativity – The Greatest Advantage of Rule Languages

- Separates between
  - *What* is the problem?
  - *How* is the problem solved?
- Built-in problem-solving
  ⇒ Allows to concentrate on problem-specification
- Add and modify rules easily
- Supports rapid prototyping and stepwise refinement
- Finding solutions where no explicit algorithm is known
- Adaption to frequently changing prerequisites
Declarativity – The Greatest Advantage of Rule Languages

- Separates between
  - *What* is the problem?
  - *How* is the problem solved?

- Built-in problem-solving
  ⇒ Allows to concentrate on problem-specification
- Add and modify rules easily
- Supports rapid prototyping and stepwise refinement
- Finding solutions where no explicit algorithm is known
- Adaption to frequently changing prerequisites
Declarativity – The Greatest Advantage of Rule Languages

- Separates between
  - *What* is the problem?
  - *How* is the problem solved?

- Built-in problem-solving
  ⇒ Allows to concentrate on problem-specification

- Add and modify rules easily
- Supports rapid prototyping and stepwise refinement
- Finding solutions where no explicit algorithm is known
- Adaptation to frequently changing prerequisites
Declarativity – The Greatest Advantage of Rule Languages

- Separates between
  - *What* is the problem?
  - *How* is the problem solved?

- Built-in problem-solving
  ⇒ Allows to concentrate on problem-specification

- Add and modify rules easily
  - Supports rapid prototyping and stepwise refinement
  - Finding solutions where no explicit algorithm is known
  - Adaption to frequently changing prerequisites
Declarativity – The Greatest Advantage of Rule Languages

- Separates between
  - What is the problem?
  - How is the problem solved?
- Built-in problem-solving
  $\Rightarrow$ Allows to concentrate on problem-specification
- Add and modify rules easily
- Supports rapid prototyping and stepwise refinement
  - Finding solutions where no explicit algorithm is known
  - Adaption to frequently changing prerequisites
Declarativity – The Greatest Advantage of Rule Languages

- Separates between
  - What is the problem?
  - How is the problem solved?
- Built-in problem-solving
  ⇒ Allows to concentrate on problem-specification
- Add and modify rules easily
- Supports rapid prototyping and stepwise refinement
- Finding solutions where no explicit algorithm is known
- Adaption to frequently changing prerequisites
Declarativity – The Greatest Advantage of Rule Languages

- Separates between
  - *What* is the problem?
  - *How* is the problem solved?

- Built-in problem-solving
  ⇒ Allows to concentrate on problem-specification

- Add and modify rules easily

- Supports rapid prototyping and stepwise refinement

- Finding solutions where no explicit algorithm is known

- Adaption to frequently changing prerequisites
An inference engine depends on

- a logical system with reasonable soundness & completeness properties
- a search method which
  - preserves (most of) these properties
  - provides an adequate degree of efficiency
An inference engine depends on

- a logical system with reasonable soundness & completeness properties
- a search method which
  - preserves (most of) these properties
  - provides an adequate degree of efficiency
Necessary Design Decisions

- tuple-oriented vs. set-oriented
- forward vs. backward reasoning
- ...

No Special Assumptions for this Paper

- Complete and space-efficient search method for rule-engines
- Particularly applicable to
  - Backward reasoning with and without memorization
  - Forward reasoning approaches with some goal guidance

Simon Brodt, François Bry, Norbert Eisinger
Necessary Design Decisions

- tuple-oriented vs. set-oriented
- forward vs. backward reasoning
- ...

No Special Assumptions for this Paper

- Complete and space-efficient search method for rule-engines
- Particularly applicable to
  - Backward reasoning with and without memorization
  - Forward reasoning approaches with some goal guidance
Necessary Design Decisions

- tuple-oriented vs. set-oriented
- forward vs. backward reasoning
- ...

No Special Assumptions for this Paper

- Complete and space-efficient search method for rule-engines
- Particularly applicable to
  - Backward reasoning with and without memorization
  - Forward reasoning approaches with some goal guidance
Necessary Design Decisions
- tuple-oriented vs. set-oriented
- forward vs. backward reasoning
- ...

No Special Assumptions for this Paper
- Complete and space-efficient search method for rule-engines
- Particularly applicable to
  - Backward reasoning with and without memorization
  - Forward reasoning approaches with some goal guidance
Not Much Choice

- Depth-First-Search (D-search)
- Breadth-First-Search (B-search)
- Iterative Deepening
- Iterative Broadening
Not Much Choice

- Depth-First-Search (D-search)
- Breadth-First-Search (B-search)
- Iterative Deepening
- Iterative Broadening
Not Much Choice

- Depth-First-Search (D-search)
- Breadth-First-Search (B-search)
- Iterative Deepening
  - Iterative Broadening
Not Much Choice

- Depth-First-Search (D-search)
- Breadth-First-Search (B-search)
- Iterative Deepening
- Iterative Broadening
Not Much Choice

- Depth-First-Search (D-search)
- Breadth-First-Search (B-search)
- Iterative Deepening
- Iterative Broadening
Not Much Choice

- Depth-First-Search (D-search)
- Breadth-First-Search (B-search)
- Iterative Deepening
- Iterative Broadening

Only uninformed search methods can be used
Not Much Choice

- Depth-First-Search (D-search)
- Breadth-First-Search (B-search)
- Iterative Deepening
- Iterative Broadening

Only uninformed search methods can be used
**Completeness** on finite and infinite search trees. Every node in the search space is visited after a finite number of steps.

Polynomial space complexity $O(d^c)$
- $c = constant$
- $d = maximum depth reached so far$
  
  (or of the entire tree, if it is finite)

Linear time complexity $O(n)$
- $n = number of nodes visited at least once$
  
  (or of the entire tree, if it is finite)
Desiderata for Search Methods

Completeness on finite and infinite search trees. Every node in the search space is visited after a finite number of steps.

Polynomial space complexity $O(d^c)$

\[ c = constant \]
\[ d = maximum \ depth \ reached \ so \ far \]
(or of the entire tree, if it is finite)

Linear time complexity $O(n)$

\[ n = number \ of \ nodes \ visited \ at \ least \ once \]
(or of the entire tree, if it is finite)
Desiderata for Search Methods

**Completeness** on finite and infinite search trees.
Every node in the search space is visited after a finite number of steps.

**Polynomial space complexity** $O(d^c)$

$c = constant$

$d = maximum depth reached so far$
(or of the entire tree, if it is finite)

**Linear time complexity** $O(n)$

$n = number of nodes visited at least once$
(or of the entire tree, if it is finite)
Traditional Methods Fail

**D-search**
Incomplete on infinite trees

**B-search**
Exponential space-complexity in the depth of the tree

**Iterative Deepening**
Frequent re-evaluation

**Iterative Broadening**
Incomplete on infinite trees
Frequent re-evaluation
Traditional Methods Fail

**D-search**
Incomplete on infinite trees

**B-search**
Exponential space-complexity in the depth of the tree

**Iterative Deepening**
Frequent re-evaluation

**Iterative Broadening**
Incomplete on infinite trees
Frequent re-evaluation
Traditional Methods Fail

**D-search**
Incomplete on infinite trees

**B-search**
Exponential space-complexity in the depth of the tree

**Iterative Deepening**
Frequent re-evaluation

**Iterative Broadening**
Incomplete on infinite trees
Frequent re-evaluation
Traditional Methods Fail

D-search
Incomplete on infinite trees

B-search
Exponential space-complexity in the depth of the tree

Iterative Deepening
Frequent re-evaluation

Iterative Broadening
Incomplete on infinite trees
Frequent re-evaluation
Sensible Compromise? (Prolog)

- Use D-search
- Give rule authors some control to avoid infinite dead ends (e.g. ordering of the rules, ...)

Declarativity gets lost
Sensible Compromise? (Prolog)

- Use D-search
- Give rule authors some control to avoid infinite dead ends (e.g. ordering of the rules, ...)

Declarativity gets lost
Term Representation for Natural Numbers

- zero represents 0
- succ(X,Y) can provide the predecessor X to any Y representing a nonzero natural number

Program

\[
\begin{align*}
\text{nat}(\text{zero}) & \leftarrow \\
\text{nat}(Y) & \leftarrow \text{succ}(X,Y) \land \text{nat}(X) \\
\text{nat}_2(X,Y) & \leftarrow \text{nat}(X) \land \text{nat}(Y) \\
\text{less}(X,Y) & \leftarrow "reasonably \ defined" \\
\end{align*}
\]
Problem 1 – Incomplete Enumerations

Program

\[
\begin{align*}
nat(\text{zero}) & \leftarrow \\
nat(Y) & \leftarrow \text{succ}(X,Y) \land \text{nat}(X) \\
nat_2(X,Y) & \leftarrow \text{nat}(X) \land \text{nat}(Y) \\
\text{less}(X,Y) & \leftarrow "reasonably\ defined"
\end{align*}
\]

Queries

1. \(\leftarrow \text{nat}(X)\)
2. \(\leftarrow \text{nat}_2(X,Y)\)

Expected Results

1. Enumeration of \(\mathbb{N}\)
2. Enumeration of \(\mathbb{N} \times \mathbb{N}\)

Prolog’s Results

1. Enumeration of \(\mathbb{N}\)
2. Enumeration of \(\{0\} \times \mathbb{N}\)
Problem 1 – Incomplete Enumerations

**Program**

\[
\begin{align*}
nat(\text{zero}) & \leftarrow \\
nat(Y) & \leftarrow \text{succ}(X,Y) \land nat(X) \\
nat_2(X,Y) & \leftarrow nat(X) \land nat(Y) \\
less(X,Y) & \leftarrow "\text{reasonably defined}"
\end{align*}
\]

**Queries**

1. \( \leftarrow nat(X) \)
2. \( \leftarrow nat_2(X,Y) \)

**Expected Results**

1. Enumeration of \( \mathbb{N} \)
2. Enumeration of \( \mathbb{N} \times \mathbb{N} \)

**Prolog’s Results**

1. Enumeration of \( \mathbb{N} \)
2. Enumeration of \( \{0\} \times \mathbb{N} \)
**Problem 1 – Incomplete Enumerations**

<table>
<thead>
<tr>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nat(zero) ←</code></td>
</tr>
<tr>
<td><code>nat(Y) ← succ(X,Y) ∧ nat(X)</code></td>
</tr>
<tr>
<td><code>nat₂(X,Y) ← nat(X) ∧ nat(Y)</code></td>
</tr>
<tr>
<td><code>less(X,Y) ← &quot;reasonably defined&quot;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ← <code>nat(X)</code></td>
</tr>
<tr>
<td>2. ← <code>nat₂(X,Y)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Enumeration of (\mathbb{N})</td>
</tr>
<tr>
<td>2. Enumeration of (\mathbb{N} \times \mathbb{N})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prolog’s Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Enumeration of (\mathbb{N})</td>
</tr>
<tr>
<td>2. Enumeration of ({0} \times \mathbb{N})</td>
</tr>
</tbody>
</table>
Problem 2 – Non-Commutativity

Program

\[
\begin{align*}
\text{nat}(\text{zero}) & \leftarrow \\
\text{nat}(Y) & \leftarrow \text{succ}(X,Y) \land \text{nat}(X) \\
\text{nat}_2(X,Y) & \leftarrow \text{nat}(X) \land \text{nat}(Y) \\
\text{less}(X,Y) & \leftarrow "reasonably defined"
\end{align*}
\]

Queries (Assume Single-Answer-Mode)

1. \(\leftarrow \text{less}(\text{zero},X) \land \text{nat}_2(X,Y)\)
2. \(\leftarrow \text{nat}_2(X,Y) \land \text{less}(\text{zero},X)\)

Expected Results

1. Yes
2. Yes

Prolog's Results

1. Yes
2. No answer (does not terminate)
Problem 2 – Non-Commutativity

Program

\[
\begin{align*}
\text{nat}(\text{zero}) & \leftarrow \\
\text{nat}(Y) & \leftarrow \text{succ}(X,Y) \land \text{nat}(X) \\
\text{nat}_2(X,Y) & \leftarrow \text{nat}(X) \land \text{nat}(Y) \\
\text{less}(X,Y) & \leftarrow "reasonably
def\text{defined}" \\
\end{align*}
\]

Queries (Assume Single-Answer-Mode)

\[
\begin{align*}
1 & \leftarrow \text{less}(\text{zero},X) \land \text{nat}_2(X,Y) \\
2 & \leftarrow \text{nat}_2(X,Y) \land \text{less}(\text{zero},X) \\
\end{align*}
\]

Expected Results

1. Yes
2. Yes

Prolog’s Results

1. Yes
2. No answer (does not terminate)
Problem 2 – Non-Commutativity

Program

\[
\begin{align*}
nat(\text{zero}) & \leftarrow \\
nat(Y) & \leftarrow \text{succ}(X,Y) \land nat(X) \\
nat_2(X,Y) & \leftarrow nat(X) \land nat(Y) \\
less(X,Y) & \leftarrow "reasonably\ defined"
\end{align*}
\]

Queries

(Assume Single-Answer-Mode)

\[
\begin{align*}
1 & \leftarrow less(\text{zero},X) \land nat_2(X,Y) \\
2 & \leftarrow nat_2(X,Y) \land less(\text{zero},X)
\end{align*}
\]

Expected Results

1. Yes
2. Yes

Prolog’s Results

1. Yes
2. No answer (does not terminate)
**Motivation**

D&B-search

Search & Partial Ordering

Conclusion

**Rule Languages & Declarativity**

Rule Languages & Search

Desiderata for Search Methods

Search & Declarativity

---

**Reason – Incomplete Search**

**SLD-resolution is fine**

Perfectly sound and complete with any literal selection function.

**Problem: Incompleteness of D-search**

The problems would not arise with a complete search method.

Choose iterative deepening?
SLD-resolution is fine
Perfectly sound and complete with any literal selection function.

Problem: Incompleteness of D-search
The problems would not arise with a complete search method

Choose iterative deepening?
Reason – Incomplete Search

SLD-resolution is fine
Perfectly sound and complete with any literal selection function.

Problem: Incompleteness of D-search
The problems would not arise with a complete search method

Choose iterative deepening?
Problem 3 – Inefficiency on Functional Rule Sets

Program

\[
\begin{align*}
\text{even}(\text{zero}) & \leftarrow \\
\text{even}(Y) & \leftarrow \text{succ}(X,Y) \land \text{odd}(X) \\
\text{odd}(Y) & \leftarrow \text{succ}(X,Y) \land \text{even}(X)
\end{align*}
\]

Query

\[
\leftarrow \text{constant}(X) \land \text{even}(X)
\]

constant(X) binds X to some fixed, large number \( n \in \mathbb{N} \).

Expected Runtime

\[O(n)\]

Runtime with Iterative-Deepening

\[O(n^2)\]

Search should not slow down the evaluation of functional rules.
Program

\begin{align*}
even(\text{zero}) & \leftarrow \\
even(Y) & \leftarrow \text{succ}(X,Y) \land \text{odd}(X) \\
odd(Y) & \leftarrow \text{succ}(X,Y) \land \text{even}(X)
\end{align*}

Query

\[ \leftarrow \text{constant}(X) \land \text{even}(X) \]

\text{constant}(X) \text{ binds } X \text{ to some fixed, large number } n \in \mathbb{N}.

Expected Runtime

\begin{align*}
\text{Runtime with Iterative-Deepening} & \\
O(n^2) & \\
\end{align*}

Search should not slow down the evaluation of functional rules.
Problem 3 – Inefficiency on Functional Rule Sets

Program

\[
\begin{align*}
\text{even}(\text{zero}) & \leftarrow \\
\text{even}(Y) & \leftarrow \text{succ}(X,Y) \land \text{odd}(X) \\
\text{odd}(Y) & \leftarrow \text{succ}(X,Y) \land \text{even}(X)
\end{align*}
\]

Query

\[
\leftarrow \text{constant}(X) \land \text{even}(X)
\]

constant\((X)\) binds \(X\) to some fixed, large number \(n \in \mathbb{N}\).

Expected Runtime

\(O(n)\)

Runtime with Iterative-Deepening

\(O(n^2)\)

Search should not slow down the evaluation of functional rules.
Problem 3 – Inefficiency on Functional Rule Sets

Program

even(zero) ←
even(Y) ← succ(X,Y) ∧ odd(X)
odd(Y) ← succ(X,Y) ∧ even(X)

Query

← constant(X) ∧ even(X)

constant(X) binds X to some fixed, large number \( n \in \mathbb{N} \).

Expected Runtime

\( O(n) \)

Runtime with Iterative-Deepening

\( O(n^2) \)

Search should not slow down the evaluation of functional rules
A New Algorithm – D&B-search

- Integrates D-search and B-search
- Complete on finite and infinite trees
- Linear space complexity in depth for basic algorithm
- Non-repetitive
- Family of algorithms in parameter $c$ with
  - Complete for $c > 0$
  - Polynomial space-requirement $O(d^c)$ in depth for $c < \infty$
  - D-search and B-search as extreme cases
- Only simple datastructures needed
- Properties are proved
A New Algorithm – D&B-search

- Integrates D-search and B-search
- Complete on finite and infinite trees
- Linear space complexity in depth for basic algorithm
- Non-repetitive
- Family of algorithms in parameter $c$ with
  - Complete for $c > 0$
  - Polynomial space-requirement $O(d^c)$ in depth for $c < \infty$
  - D-search and B-search as extreme cases
- Only simple datastructures needed
- Properties are proved
Overview

1. D&B-search
2. Search & Partial Ordering
3. Conclusion
D&B-search

1. D&B-search
   - The Basic Algorithm
   - The D&B-Family

2. Search & Partial Ordering

3. Conclusion
D-search starts
- D-search passes depth bound $f_0$
- B-search completes level 0 (no work to do)
- D-search passes depth bound $f_1$
- B-search completes level 1
- D-search passes depth bound $f_2$
- B-search completes level 2
- D-search starts
- D-search passes depth bound $f_0$
- B-search completes level 0 (no work to do)
- D-search passes depth bound $f_1$
- B-search completes level 1
- D-search passes depth bound $f_2$
- B-search completes level 2
D-search starts
- D-search passes depth bound $f_0$
- B-search completes level 0 (no work to do)
- D-search passes depth bound $f_1$
- B-search completes level 1
- D-search passes depth bound $f_2$
- B-search completes level 2
D-search starts
D-search passes depth bound $f_0$
B-search completes level 0 (no work to do)
D-search passes depth bound $f_1$
B-search completes level 1
D-search passes depth bound $f_2$
B-search completes level 2
- D-search starts
- D-search passes depth bound $f_0$
- B-search completes level 0 (no work to do)
- D-search passes depth bound $f_1$
- B-search completes level 1
- D-search passes depth bound $f_2$
- B-search completes level 2
D-search starts

- D-search passes depth bound $f_0$
- B-search completes level 0 (no work to do)
- D-search passes depth bound $f_1$
- B-search completes level 1
- D-search passes depth bound $f_2$
- B-search completes level 2
The Basic Algorithm

The D&B-Family

D-search starts
D-search passes depth bound $f_0$
B-search completes level 0 (no work to do)
D-search passes depth bound $f_1$
B-search completes level 1
D-search passes depth bound $f_2$
B-search completes level 2
The Basic Algorithm

1. D-search starts
2. D-search passes depth bound $f_0$
3. B-search completes level 0 (no work to do)
4. D-search passes depth bound $f_1$
5. B-search completes level 1
6. D-search passes depth bound $f_2$
7. B-search completes level 2

**Generally**

- D-search passes depth bound $f_{i+1}$ only if the level $i$ has been completed
- B-search completes the level $i$ only if depth bound $f_i$ has been passed
D-search starts
D-search passes depth bound $f_0$
B-search completes level 0
(no work to do)
D-search passes depth bound $f_1$
B-search completes level 1
D-search passes depth bound $f_2$
B-search completes level 2

Generally
- D-search passes depth bound $f_{i+1}$ only if the level $i$ has been completed
- B-search completes the level $i$ only if depth bound $f_i$ has been passed
D-search starts
D-search passes depth bound $f_0$
B-search completes level 0
(no work to do)
D-search passes depth bound $f_1$
B-search completes level 1
D-search passes depth bound $f_2$
B-search completes level 2

Generally
D-search passes depth bound $f_{i+1}$ only if the level $i$ has been completed
B-search completes the level $i$ only if depth bound $f_i$ has been passed
Observations

- D-search advances exponentially faster than B-search
- The number of nodes to be stored is only polynomial in the maximum depth (if \( f_i \) is exponential in \( i \))
The Basic Algorithm

D-search starts
D-search passes depth bound $f_0$
B-search completes level 0 (no work to do)
D-search passes depth bound $f_1$
B-search completes level 1
D-search passes depth bound $f_2$
B-search completes level 2

Observations
- D-search advances exponentially faster than B-search
- The number of nodes to be stored is only polynomial in the maximum depth (if $f_i$ is exponential in $i$)
Observations

- D-search advances exponentially faster than B-search
- The number of nodes to be stored is only polynomial in the maximum depth (if $f_i$ is exponential in $i$)
D&B-search – Idea

- Alternate D-search with B-search
- Rotation is controlled by a sequence $f_0, f_1, f_2, \ldots$ of depth bounds
  - Defined by a function $\mathbb{N} \rightarrow \mathbb{N}, \ i \mapsto f_i$
  - $i < f_i < f_{i+1}$
- $f_i = 2^i$ for the examples
A node is “earlier” than another if (unrestricted) D-search would expand it first

- **Pivot-node** \( s_i \): earliest node at depth \( f_i \)
- **Pre-pivot-set** \( S_0 \): nodes earlier than \( s_0 \)
- **\( D_i \)**: nodes earlier than \( s_{i+1} \)
- **\( B_i \)**: nodes at depth \( i \)
- **Inter-pivot-set** \( S_{i+1} = (D_i \cup B_i) \setminus X_i \)
  - is expanded in-between \( s_i \) and \( s_{i+1} \)
- **\( X_i = S_0 \cup s_0 \cup \ldots \cup S_i \cup s_i \)**
- **Post-pivot-set** \( R \): the rest of the nodes
A node is “earlier” than another if (unrestricted) D-search would expand it first:

- **Pivot-node** $s_i$: earliest node at depth $f_i$
- **Pre-pivot-set** $S_0$: nodes earlier than $s_0$
- $D_i$: nodes earlier than $s_{i+1}$
- $B_i$: nodes at depth $i$
- **Inter-pivot-set** $S_{i+1} = (D_i \cup B_i) \setminus X_i$ is expanded in-between $s_i$ and $s_{i+1}$
- $X_i = S_0 \cup s_0 \cup \ldots \cup S_i \cup s_i$
- **Post-pivot-set** $R$: the rest of the nodes
A node is “earlier” than another if (unrestricted) D-search would expand it first

- **Pivot-node** $s_i$: earliest node at depth $f_i$
- **Pre-pivot-set** $S_0$: nodes earlier than $s_0$
- $D_i$: nodes earlier than $s_{i+1}$
- $B_i$: nodes at depth $i$
- **Inter-pivot-set** $S_{i+1} = (D_i \cup B_i) \setminus X_i$
- $X_i = S_0 \cup s_0 \cup \ldots \cup S_i \cup s_i$
- **Post-pivot-set** $R$: the rest of the nodes
A node is “earlier” than another if (unrestricted) D-search would expand it first

- **Pivot-node** $s_i$: earliest node at depth $f_i$
- **Pre-pivot-set** $S_0$: nodes earlier than $s_0$
- **$D_i$**: nodes earlier than $s_{i+1}$
- **$B_i$**: nodes at depth $i$
- **Inter-pivot-set** $S_{i+1} = (D_i \cup B_i) \setminus X_i$ is expanded in-between $s_i$ and $s_{i+1}$
- **$X_i = S_0 \cup s_0 \cup \ldots \cup S_i \cup s_i$**
- **Post-pivot-set** $R$: the rest of the nodes
A node is “earlier” than another if (unrestricted) D-search would expand it first

- **Pivot-node** $s_i$: earliest node at depth $f_i$
- **Pre-pivot-set** $S_0$: nodes earlier than $s_0$
- **$D_i$**: nodes earlier than $s_{i+1}$
- **$B_i$**: nodes at depth $i$
- **Inter-pivot-set** $S_{i+1} = (D_i \cup B_i) \setminus X_i$ is expanded in-between $s_i$ and $s_{i+1}$
- **$X_i = S_0 \cup s_0 \cup \ldots \cup S_i \cup s_i$**
- **Post-pivot-set** $R$: the rest of the nodes
A node is “earlier” than another if (unrestricted) D-search would expand it first

- **Pivot-node** \( s_i \): earliest node at depth \( f_i \)
- **Pre-pivot-set** \( S_0 \): nodes earlier than \( s_0 \)
- **\( D_i \)**: nodes earlier than \( s_{i+1} \)
- **\( B_i \)**: nodes at depth \( i \)
- **Inter-pivot-set** \( S_{i+1} = (D_i \cup B_i) \setminus X_i \)
  - is expanded in-between \( s_i \) and \( s_{i+1} \)
- \( X_i = S_0 \cup s_0 \cup \ldots \cup S_i \cup s_i \)
- **Post-pivot-set** \( R \): the rest of the nodes
A node is “earlier” than another if (unrestricted) D-search would expand it first
- *Pivot-node* $s_i$: earliest node at depth $f_i$
- *Pre-pivot-set* $S_0$: nodes earlier than $s_0$
- $D_i$: nodes earlier than $s_{i+1}$
- $B_i$: nodes at depth $i$
- *Inter-pivot-set* $S_{i+1} = (D_i \cup B_i) \setminus X_i$
  - is expanded in-between $s_i$ and $s_{i+1}$
- $X_i = S_0 \cup s_0 \cup \ldots \cup S_i \cup s_i$
- *Post-pivot-set* $R$: the rest of the nodes
D&B-search – Complete Infinite Tree

- D-search expands all nodes in $S_0$
- D-search passes $s_0$
- $S_1$ is finished
- D-search passes $s_1$
- B-search expands the rest of $B_1$
- $S_2$ is finished
- D-search passes $s_2$
- B-search expands the rest of $B_2$
- $S_3$ is finished
**D&B-search – Complete Infinite Tree**

- D-search expands all nodes in $S_0$
- D-search passes $s_0$
- $S_1$ is finished
- D-search passes $s_1$
- B-search expands the rest of $B_1$
- $S_2$ is finished
- D-search passes $s_2$
- B-search expands the rest of $B_2$
- $S_3$ is finished
D&B-search – Complete Infinite Tree

- D-search expands all nodes in $S_0$
- D-search passes $s_0$
- $S_1$ is finished
  - D-search passes $s_1$
  - B-search expands the rest of $B_1$
  - $S_2$ is finished
  - D-search passes $s_2$
  - B-search expands the rest of $B_2$
  - $S_3$ is finished
D&B-search – Complete Infinite Tree

- D-search expands all nodes in $S_0$
- D-search passes $s_0$
- $S_1$ is finished
- D-search passes $s_1$
- B-search expands the rest of $B_1$
- $S_2$ is finished
- D-search passes $s_2$
- B-search expands the rest of $B_2$
- $S_3$ is finished
D&B-search – Complete Infinite Tree

- D-search expands all nodes in $S_0$
- D-search passes $s_0$
- $S_1$ is finished
- D-search passes $s_1$
- B-search expands the rest of $B_1$
- $S_2$ is finished
- D-search passes $s_2$
- B-search expands the rest of $B_2$
- $S_3$ is finished
D&B-search – Complete Infinite Tree

- D-search expands all nodes in $S_0$
- D-search passes $s_0$
- $S_1$ is finished
- D-search passes $s_1$
- B-search expands the rest of $B_1$
- $S_2$ is finished
- D-search passes $s_2$
- B-search expands the rest of $B_2$
- $S_3$ is finished
D&B-search – Complete Infinite Tree

- D-search expands all nodes in $S_0$
- D-search passes $s_0$
- $S_1$ is finished
- D-search passes $s_1$
- B-search expands the rest of $B_1$
- $S_2$ is finished
- D-search passes $s_2$
- B-search expands the rest of $B_2$
- $S_3$ is finished
D&B-search – Complete Infinite Tree

- D-search expands all nodes in $S_0$
- D-search passes $s_0$
- $S_1$ is finished
- D-search passes $s_1$
- B-search expands the rest of $B_1$
- $S_2$ is finished
- D-search passes $s_2$
- B-search expands the rest of $B_2$
- $S_3$ is finished
D&B-search – Complete Infinite Tree

- D-search expands all nodes in $S_0$
- D-search passes $s_0$
- $S_1$ is finished
- D-search passes $s_1$
- B-search expands the rest of $B_1$
- $S_2$ is finished
- D-search passes $s_2$
- B-search expands the rest of $B_2$
- $S_3$ is finished
D&B-search – Complete Infinite Tree

- D-search expands all nodes in $S_0$
- D-search passes $s_0$
- $S_1$ is finished
- D-search passes $s_1$
- B-search expands the rest of $B_1$
- $S_2$ is finished
- D-search passes $s_2$
- B-search expands the rest of $B_2$
- $S_3$ is finished

Observation

D&B-search expands the nodes in the order
$S_0, s_0, S_1, s_1, \ldots, S_i, s_i, \ldots, R$
D&B-search – Finite Tree

- $S_2$ is finished
- D-search expands $s_2$
- D-search reaches the max. depth in $R$
  (no $s_3$ in this tree)
- B-search may complete $B_2$
- D-search continues $R$
- D-search continues $R$
- D-search finishes $R$
- Search is finished
D&B-search – Finite Tree

- $S_2$ is finished
- D-search expands $s_2$
- D-search reaches the max. depth in $R$ (no $s_3$ in this tree)
- B-search may complete $B_2$
- D-search continues $R$
- D-search continues $R$
- D-search finishes $R$
- Search is finished
D&B-search – Finite Tree

- \( S_2 \) is finished
- D-search expands \( s_2 \)
- D-search reaches the max. depth in \( R \) (no \( s_3 \) in this tree)
- B-search may complete \( B_2 \)
- D-search continues \( R \)
- D-search continues \( R \)
- D-search finishes \( R \)
- Search is finished
D&B-search – Finite Tree

- $S_2$ is finished
- D-search expands $s_2$
- D-search reaches the max. depth in $R$ (no $s_3$ in this tree)
- B-search may complete $B_2$
- D-search continues $R$
- D-search continues $R$
- D-search finishes $R$
- Search is finished

Observation
B-search stops shortly after D-search reaches the max. depth
D&B-search – Finite Tree

- $S_2$ is finished
- D-search expands $s_2$
- D-search reaches the max. depth in $R$
  (no $s_3$ in this tree)
- B-search may complete $B_2$
- D-search continues $R$
- D-search continues $R$
- D-search finishes $R$
- Search is finished
**D&B-search – Finite Tree**

- $S_2$ is finished
- D-search expands $s_2$
- D-search reaches the max. depth in $R$ (no $s_3$ in this tree)
- B-search may complete $B_2$
- D-search continues $R$
- D-search continues $R$
- D-search finishes $R$
- Search is finished
D&B-search – Finite Tree

- $S_2$ is finished
- D-search expands $s_2$
- D-search reaches the max. depth in $R$ (no $s_3$ in this tree)
- B-search may complete $B_2$
- D-search continues $R$
- D-search continues $R$
- D-search finishes $R$
- Search is finished

Observation
- B-search stops shortly after D-search reaches the max. depth
- Most of the tree is expanded by D-search
D&B-search – Finite Tree

- $S_2$ is finished
- D-search expands $s_2$
- D-search reaches the max. depth in $R$
  (no $s_3$ in this tree)
- B-search may complete $B_2$
- D-search continues $R$
- D-search continues $R$
- D-search finishes $R$
- Search is finished

**Observation**

- B-search stops shortly after D-search reaches the max. depth
- Most of the tree is expanded by D-search
D&B-search – Finite Tree

- $S_2$ is finished
- D-search expands $s_2$
- D-search reaches the max. depth in $R$
  (no $s_3$ in this tree)
- B-search may complete $B_2$
- D-search continues $R$
- D-search continues $R$
- D-search finishes $R$
- Search is finished

Observation

- B-search stops shortly after D-search reaches the max. depth
- Most of the tree is expanded by D-search
D&B-search – Non-Complete Infinite Tree

- $S_1$ is finished
- D-search passes $s_1$
  - B-search may complete $B_1$
  - D-search finishes $S_2$
  - D-search passes $s_2$
  - B-search completes $B_2$
- $S_3$ is finished
- D-search passes $s_3$
- B-search completes $B_3$
D&B-search – Non-Complete Infinite Tree

- $S_1$ is finished
- D-search passes $s_1$
- B-search may complete $B_1$
- D-search finishes $S_2$
- D-search passes $s_2$
- B-search completes $B_2$
- $S_3$ is finished
- D-search passes $s_3$
- B-search completes $B_3$
D&B-search – Non-Complete Infinite Tree

- $S_1$ is finished
- D-search passes $s_1$
- B-search may complete $B_1$
- D-search finishes $S_2$
  - D-search passes $s_2$
  - B-search completes $B_2$
- $S_3$ is finished
  - D-search passes $s_3$
  - B-search completes $B_3$

Observation
D-search "vanishes" in the earliest infinite branch
Most of the tree is expanded by B-search
D&B-search – Non-Complete Infinite Tree

- $S_1$ is finished
- D-search passes $s_1$
- B-search may complete $B_1$
- D-search finishes $S_2$
- D-search passes $s_2$
- B-search completes $B_2$
- $S_3$ is finished
- D-search passes $s_3$
- B-search completes $B_3$
D&B-search – Non-Complete Infinite Tree

- $S_1$ is finished
- D-search passes $s_1$
- B-search may complete $B_1$
- D-search finishes $S_2$
- D-search passes $s_2$
- B-search completes $B_2$
- $S_3$ is finished
- D-search passes $s_3$
- B-search completes $B_3$
D&B-search – Non-Complete Infinite Tree

- $S_1$ is finished
- D-search passes $s_1$
- B-search may complete $B_1$
- D-search finishes $S_2$
- D-search passes $s_2$
- B-search completes $B_2$
- $S_3$ is finished
- D-search passes $s_3$
- B-search completes $B_3$
**D&B-search – Non-Complete Infinite Tree**

- $S_1$ is finished
- D-search passes $s_1$
- B-search may complete $B_1$
- D-search finishes $S_2$
- D-search passes $s_2$
- B-search completes $B_2$
- $S_3$ is finished
- D-search passes $s_3$
- B-search completes $B_3$
D&B-search – Non-Complete Infinite Tree

- $S_1$ is finished
- D-search passes $s_1$
- B-search may complete $B_1$
- D-search finishes $S_2$
- D-search passes $s_2$
- B-search completes $B_2$
- $S_3$ is finished
- D-search passes $s_3$
- B-search completes $B_3$

Observation

- D-search “vanishes” in the earliest infinite branch
- Most of the tree is expanded by B-search
Observation
- D-search “vanishes” in the earliest infinite branch
- Most of the tree is expanded by B-search
D&B-search – Non-Complete Infinite Tree

- $S_1$ is finished
- D-search passes $s_1$
- B-search may complete $B_1$
- D-search finishes $S_2$
- D-search passes $s_2$
- B-search completes $B_2$
- $S_3$ is finished
- D-search passes $s_3$
- B-search completes $B_3$

Observation

- D-search “vanishes” in the earliest infinite branch
- Most of the tree is expanded by B-search
D&B-search – Adaptivity

D&B-Search
- behaves almost like D-search on finite trees
- behaves almost like B-search on infinite trees
→ has a kind of built-in adaptivity
behaves like the “best” uninformed search method for the tree

Similar effect when D-search and iterative-deepening are combined
D&B-search – Adapitivity

D&B-Search

- behaves almost like D-search on finite trees
- behaves almost like B-search on infinite trees

⇒ has a kind of built-in adaptivity
behaves like the “best” uninformed search method for the tree

Similar effect when D-search and iterative-deepening are combined
D&B-search – Adaptivity

D&B-Search

- behaves almost like D-search on finite trees
- behaves almost like B-search on infinite trees

⇒ **has a kind of built-in adaptivity**

behaves like the “best” uninformed search method for the tree

Similar effect when D-search and iterative-deepening are combined
D&B-search – Adaptness

D&B-Search

- behaves almost like D-search on finite trees
- behaves almost like B-search on infinite trees

⇒ **has a kind of built-in adaptivity**
behaves like the “best” uninformed search method for the tree

Similar effect when D-search and iterative-deepening are combined
Assume that the tree’s branching factor is bounded by $b \in \mathbb{N}$

- Parameterise the function $f_i$ with $c \in \mathbb{N} \cup \{\infty\}$
- Idea: $f_{c,i} := \lfloor b^\frac{i}{c} \rfloor$
- To get monotonicity: $f_{c,i} := \lfloor b^\frac{i}{c} \rfloor + i$
Assume that the tree’s branching factor is bounded by $b \in \mathbb{N}$

- Parameterise the function $f_i$ with $c \in \mathbb{N} \cup \{\infty\}$
- Idea: $f_{c,i} := \lfloor b_i^c \rfloor$
- To get monotonicity: $f_{c,i} := \lfloor b_i^c \rfloor + i$
Assume that the tree’s branching factor is bounded by $b \in \mathbb{N}$

- Parameterise the function $f_i$ with $c \in \mathbb{N} \cup \{\infty\}$
- Idea: $f_{c,i} := \lfloor b^i c \rfloor$
- To get monotonicity: $f_{c,i} := \lfloor b^i c \rfloor + i$
The D&B-Family

\[ f_{c,i} := \lfloor b^i_c \rfloor + i \]

**Properties**

- For \( 1 \leq c \leq \infty \) the algorithm is complete (for \( c = 0 \) it is not)
- For \( 1 \leq c < \infty \) its space complexity is \( O(d^c) \)
- For \( c = 0 \) it corresponds to D-search because \( f_{0,0} = \infty \).
  The pre-pivot-set \( S_0 \) contains all nodes of the whole tree.
- For \( c = \infty \) it corresponds to B-search because \( f_{\infty,i} = i + 1 \).
  All sets \( D_i \setminus X_i \) are empty, thus \( S_{i+1} = B_i \setminus \{s_i\} \)
The D&B-Family

\[ f_{c,i} := \lfloor b^i_c \rfloor + i \]

**Properties**

- For \( 1 \leq c \leq \infty \) the algorithm is complete (for \( c = 0 \) it is not).
- For \( 1 \leq c < \infty \) its space complexity is \( O(d^c) \).
- For \( c = 0 \) it corresponds to D-search because \( f_{0,0} = \infty \).
  The pre-pivot-set \( S_0 \) contains all nodes of the whole tree.
- For \( c = \infty \) it corresponds to B-search because \( f_{\infty,i} = i + 1 \).
  All sets \( D_i \setminus X_i \) are empty, thus \( S_{i+1} = B_i \setminus \{s_i\} \).
The D&B-Family

\[ f_{c,i} := \left\lfloor b_i^c \right\rfloor + i \]

**Properties**

- For \( 1 \leq c \leq \infty \) the algorithm is complete (for \( c = 0 \) it is not).
- For \( 1 \leq c < \infty \) its space complexity is \( O(d^c) \).
- For \( c = 0 \) it corresponds to D-search because \( f_{0,0} = \infty \).
  The pre-pivot-set \( S_0 \) contains all nodes of the whole tree.
- For \( c = \infty \) it corresponds to B-search because \( f_{\infty,i} = i + 1 \).
  All sets \( D_i \setminus X_i \) are empty, thus \( S_{i+1} = B_i \setminus \{s_i\} \).
The D&B-Family

\[ f_{c,i} := \lfloor b^i \rfloor + i \]

Properties

- For \( 1 \leq c \leq \infty \) the algorithm is complete (for \( c = 0 \) it is not).
- For \( 1 \leq c < \infty \) its space complexity is \( O(d^c) \).
- For \( c = 0 \) it corresponds to D-search because \( f_{0,0} = \infty \).
  The pre-pivot-set \( S_0 \) contains all nodes of the whole tree.
- For \( c = \infty \) it corresponds to B-search because \( f_{\infty,i} = i + 1 \).
  All sets \( D_i \setminus X_i \) are empty, thus \( S_{i+1} = B_i \setminus \{s_i\} \).
The D&B-Family

Advantages

- $c$ expresses how much memory is invested in completeness
- Almost arbitrary gradation between the two extremes D-search ($c = 0$) and B-search ($c = \infty$)
- Space complexity polynomial in depth
- Time complexity linear in size
- $c$ can be used as parameter for a single implementation
- $c$ may be adapted even during the traversal
Advantages

- $c$ expresses how much memory is invested in completeness
- Almost arbitrary gradation between the two extremes D-search ($c = 0$) and B-search ($c = \infty$)
- Space complexity polynomial in depth
- Time complexity linear in size
- $c$ can be used as parameter for a single implementation
- $c$ may be adapted even during the traversal
The D&B-Family

Advantages

- \( c \) expresses how much memory is invested in completeness
- Almost arbitrary gradation between the two extremes: D-search \((c = 0)\) and B-search \((c = \infty)\)
- Space complexity polynomial in depth
- Time complexity linear in size
- \( c \) can be used as parameter for a single implementation
- \( c \) may be adapted even during the traversal
The D&B-Family

Advantages

- $c$ expresses how much memory is invested in completeness
- Almost arbitrary gradation between the two extremes D-search ($c = 0$) and B-search ($c = \infty$)
- Space complexity polynomial in depth
- Time complexity linear in size
- $c$ can be used as parameter for a single implementation
- $c$ may be adapted even during the traversal
Advantages

- $c$ expresses how much memory is invested in completeness
- Almost arbitrary gradation between the two extremes
  - D-search ($c = 0$) and B-search ($c = \infty$)
- Space complexity polynomial in depth
- Time complexity linear in size
- $c$ can be used as parameter for a single implementation
- $c$ may be adapted even during the traversal
The D&B-Family

Advantages

- $c$ expresses how much memory is invested in completeness
- Almost arbitrary gradation between the two extremes D-search ($c = 0$) and B-search ($c = \infty$)
- Space complexity polynomial in depth
- Time complexity linear in size
- $c$ can be used as parameter for a single implementation
- $c$ may be adapted even during the traversal
Search & Partial Ordering

1. D&B-search

2. Search & Partial Ordering

3. Conclusion
Search & Partial Ordering

- Transforms problems on search algorithms to problems on partial orderings
  - Idea: Nodes ordered by their first occurrence
  - Partial orderings are a well-studied field
    - precise notation
    - Powerful instruments for proofs
      (e.g. the arithmetic for ordinal numbers)
  - Powerful characterization of completeness
  - Finite and infinite trees are covered uniformly
Search & Partial Ordering

- Transforms problems on search algorithms to problems on partial orderings
- Idea: Nodes ordered by their first occurrence
  - Partial orderings are a well-studied field
    - Precise notation
    - Powerful instruments for proofs
      (e.g. the arithmetic for ordinal numbers)
  - Powerful characterization of completeness
  - Finite and infinite trees are covered uniformly
Search & Partial Ordering

- Transforms problems on search algorithms to problems on partial orderings
- Idea: Nodes ordered by their first occurrence
- Partial orderings are a well-studied field
  - precise notation
  - Powerful instruments for proofs
    (e.g. the arithmetic for ordinal numbers)
- Powerful characterization of completeness
- Finite and infinite trees are covered uniformly
Search & Partial Ordering

- Transforms problems on search algorithms to problems on partial orderings
- Idea: Nodes ordered by their first occurrence
- Partial orderings are a well-studied field
  - precise notation
  - Powerful instruments for proofs
    (e.g. the arithmetic for ordinal numbers)
- Powerful characterization of completeness
- Finite and infinite trees are covered uniformly
Search & Partial Ordering

- Transforms problems on search algorithms to problems on partial orderings
- Idea: Nodes ordered by their first occurrence
- Partial orderings are a well-studied field
  - precise notation
  - Powerful instruments for proofs
    (e.g. the arithmetic for ordinal numbers)
- Powerful characterization of completeness
- Finite and infinite trees are covered uniformly
Characterization of Completeness

A search algorithm is complete iff for each depth \( i \) there is a depth \( f_{i+1} > i \) so that none of the nodes at depth \( f_{i+1} \) is expanded before every node at depth \( i \) has been expanded.
Characterization of Completeness

A search algorithm is complete iff for each depth \( i \) there is a depth \( f_{i+1} > i \) so that none of the nodes at depth \( f_{i+1} \) is expanded before every node at depth \( i \) has been expanded.
Characterization of Completeness

A search algorithm is complete iff for each depth $i$ there is a depth $f_{i+1} > i$ so that none of the nodes at depth $f_{i+1}$ is expanded before every node at depth $i$ has been expanded.
Characterization of Completeness

A search algorithm is complete iff for each depth $i$ there is a depth $f_{i+1} > i$ so that none of the nodes at depth $f_{i+1}$ is expanded before every node at depth $i$ has been expanded.
Characterization of Completeness

A search algorithm is complete iff for each depth $i$ there is a depth $f_{i+1} > i$ so that none of the nodes at depth $f_{i+1}$ is expanded before every node at depth $i$ has been expanded.
Characterization of Completeness

A search algorithm is complete iff for each depth $i$ there is a depth $f_{i+1} > i$ so that none of the nodes at depth $f_{i+1}$ is expanded before every node at depth $i$ has been expanded.

D&B-search

- $B_i \subseteq S_{i+1} \cup X_i$
- $S_{i+1} \cup X_i$ is completed before $s_i$, the first node at depth $f_{i+1}$

$\Rightarrow$ D&B-search is complete
Characterization of Completeness

A search algorithm is complete iff for each depth $i$ there is a depth $f_{i+1} > i$ so that none of the nodes at depth $f_{i+1}$ is expanded before every node at depth $i$ has been expanded.

D&B-search

- $B_i \subseteq S_{i+1} \cup X_i$
- $S_{i+1} \cup X_i$ is completed before $s_i$, the first node at depth $f_{i+1}$

$\Rightarrow$ D&B-search is complete
A search algorithm is complete iff for each depth $i$ there is a depth $f_{i+1} > i$ so that none of the nodes at depth $f_{i+1}$ is expanded before every node at depth $i$ has been expanded.
Characterization of Completeness

A search algorithm is complete iff for each depth $i$ there is a depth $f_{i+1} > i$ so that none of the nodes at depth $f_{i+1}$ is expanded before every node at depth $i$ has been expanded.

D&B-search

- $B_i \subseteq S_{i+1} \cup X_i$
- $S_{i+1} \cup X_i$ is completed before $s_i$, the first node at depth $f_{i+1}$

$\Rightarrow$ D&B-search is complete
Conclusion

1. D&B-search
2. Search & Partial Ordering
3. Conclusion
Novel Search method: Integrating D-search and B-search
- Ratio of D-search and B-search balanced by a parameter
- Family of algorithms in parameter $c$
  - D-search and B-Search as borderline cases
  - Complete in all non-borderline cases
  - Non-repetitive, i.e. time complexity is linear in size
  - Space complexity is polynomial in depth. Polynomial depends on parameter $c$
- Formal proofs of these properties
- Built-in adaption to the searched tree
- Better than running D-Search and B-Search in parallel
- Implementation in form of detailed pseudo-code
  - only simple datastructures needed
**D&B-search**

- Novel Search method: Integrating D-search and B-search
- Ratio of D-search and B-search balanced by a parameter
  - Family of algorithms in parameter $c$
    - D-search and B-Search as borderline cases
    - Complete in all non-borderline cases
    - Non-repetitive, i.e. time complexity is linear in size
    - Space complexity is polynomial in depth.
      Polynomial depends on parameter $c$
- Formal proofs of these properties
- Built-in adaption to the searched tree
- Better than running D-Search and B-Search in parallel
- Implementation in form of detailed pseudo-code
  → only simple datastructures needed

Simon Brodt, François Bry, Norbert Eisinger
D&B-search

- Novel Search method: Integrating D-search and B-search
- Ratio of D-search and B-search balanced by a parameter
- Family of algorithms in parameter $c$
  - D-search and B-Search as borderline cases
  - Complete in all non-borderline cases
  - Non-repetitive, i.e. time complexity is linear in size
  - Space complexity is polynomial in depth.
    Polynomial depends on parameter $c$
- Formal proofs of these properties
- Built-in adaption to the searched tree
- Better than running D-Search and B-Search in parallel
- Implementation in form of detailed pseudo-code
  → only simple datastructures needed
**D&B-search**

- Novel Search method: Integrating D-search and B-search
- Ratio of D-search and B-search balanced by a parameter $c$
- Family of algorithms in parameter $c$
  - D-search and B-Search as borderline cases
  - Complete in all non-borderline cases
  - Non-repetitive, i.e. time complexity is linear in size
  - Space complexity is polynomial in depth. Polynomial depends on parameter $c$
- Formal proofs of these properties
  - Built-in adaption to the searched tree
  - Better than running D-Search and B-Search in parallel
  - Implementation in form of detailed pseudo-code
    → only simple datastructures needed
D&B-search

- Novel Search method: Integrating D-search and B-search
- Ratio of D-search and B-search balanced by a parameter
- Family of algorithms in parameter $c$
  - D-search and B-Search as borderline cases
  - Complete in all non-borderline cases
  - Non-repetitive, i.e. time complexity is linear in size
  - Space complexity is polynomial in depth.
    Polynomial depends on parameter $c$
- Formal proofs of these properties
- Built-in adaption to the searched tree
  - Better than running D-Search and B-Search in parallel
  - Implementation in form of detailed pseudo-code
    → only simple datastructures needed
D&B-search

- Novel Search method: Integrating D-search and B-search
- Ratio of D-search and B-search balanced by a parameter
- Family of algorithms in parameter $c$
  - D-search and B-Search as borderline cases
  - Complete in all non-borderline cases
  - Non-repetitive, i.e. time complexity is linear in size
  - Space complexity is polynomial in depth.
    Polynomial depends on parameter $c$

- Formal proofs of these properties
- Built-in adaption to the searched tree
- Better than running D-Search and B-Search in parallel
- Implementation in form of detailed pseudo-code
  $\rightarrow$ only simple datastructures needed
D&B-search

- Novel Search method: Integrating D-search and B-search
- Ratio of D-search and B-search balanced by a parameter $c$
- Family of algorithms in parameter $c$
  - D-search and B-Search as borderline cases
  - Complete in all non-borderline cases
  - Non-repetitive, i.e. time complexity is linear in size
  - Space complexity is polynomial in depth.
    Polynomial depends on parameter $c$
- Formal proofs of these properties
- Built-in adaption to the searched tree
- Better than running D-Search and B-Search in parallel
- Implementation in form of detailed pseudo-code
  $\rightarrow$ only simple datastructures needed
Theoretical-Framework

- Based on partial orderings
- Covers finite and infinite trees uniformly
- High analytic power, concise and precise proofs
Future work

- Combine D-search and iterative deepening to D&I-search by the same principle
  - Behaves (almost) like D-search on finite trees
  - Behaves (almost) like iterative-deepening on infinite trees
  - Achieved by the same depth bounds $f_i$ as for D&B-search

- Same for other combinations

- Prototype implementation

- Empirical comparison to other uninformed search methods
  - Focus: Logic programming applications using backward reasoning approaches with and without memorization
Thank You