

TREES AND PROCESSES

Instituttseminar, II, UiB - vår 2006

Herman Ruge Jervell

LNS, Ifi, UiO

May 4, 2006

Abstract

Abstract

- States in processes are often described by finite trees and properties of processes – as correctness, termination, fairness, . . . – are described by properties of sequences of finite trees.

Abstract

- States in processes are often described by finite trees and properties of processes – as correctness, termination, fairness, . . . – are described by properties of sequences of finite trees.
- Highlights

Abstract

- States in processes are often described by finite trees and properties of processes – as correctness, termination, fairness, . . . – are described by properties of sequences of finite trees.
- Highlights
 - Higman's lemma

Abstract

- States in processes are often described by finite trees and properties of processes – as correctness, termination, fairness, . . . – are described by properties of sequences of finite trees.
- Highlights
 - Higman's lemma
 - Kruskal's theorem

Abstract

- States in processes are often described by finite trees and properties of processes – as correctness, termination, fairness, . . . – are described by properties of sequences of finite trees.
- Highlights
 - Higman's lemma
 - Kruskal's theorem
 - Kruskal's gap-theorem

Abstract

- States in processes are often described by finite trees and properties of processes – as correctness, termination, fairness, . . . – are described by properties of sequences of finite trees.
- Highlights
 - Higman's lemma
 - Kruskal's theorem
 - Kruskal's gap-theorem
 - Graph minor theorem

Abstract

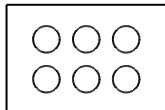
- States in processes are often described by finite trees and properties of processes – as correctness, termination, fairness, . . . – are described by properties of sequences of finite trees.
- Highlights
 - Higman's lemma
 - Kruskal's theorem
 - Kruskal's gap-theorem
 - Graph minor theorem
- Parallel development in graph theory and proof theory

A VERY SIMPLE EXAMPLE

- Situation: One bucket with balls
- Step: Take away some balls from the bucket
- Termination: No balls left

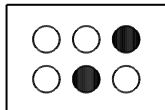
A VERY SIMPLE EXAMPLE

- Situation: One bucket with balls
- Step: Take away some balls from the bucket
- Termination: No balls left



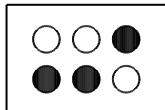
A VERY SIMPLE EXAMPLE

- Situation: One bucket with balls
- Step: Take away some balls from the bucket
- Termination: No balls left



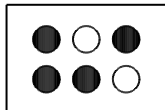
A VERY SIMPLE EXAMPLE

- Situation: One bucket with balls
- Step: Take away some balls from the bucket
- Termination: No balls left



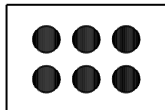
A VERY SIMPLE EXAMPLE

- Situation: One bucket with balls
- Step: Take away some balls from the bucket
- Termination: No balls left



A VERY SIMPLE EXAMPLE

- Situation: One bucket with balls
- Step: Take away some balls from the bucket
- Termination: No balls left



A VERY SIMPLE EXAMPLE

A complicated explanation

- To each state is assigned a unary tree

A VERY SIMPLE EXAMPLE

A complicated explanation

- To each state is assigned a unary tree
- We get a sequence of unary trees U_0, U_1, U_2, \dots

A VERY SIMPLE EXAMPLE

A complicated explanation

- To each state is assigned a unary tree
- We get a sequence of unary trees U_0, U_1, U_2, \dots
- If the sequence is infinite there must be $i < j$ such that U_i can be embedded in U_j

A VERY SIMPLE EXAMPLE

A complicated explanation

- To each state is assigned a unary tree
- We get a sequence of unary trees U_0, U_1, U_2, \dots
- If the sequence is infinite there must be $i < j$ such that U_i can be embedded in U_j
- This is impossible and therefore the process must terminate

THE FIRST INFINITE TREE

The tree



has the property

- It can be embedded in any tree which is not purely unary

It is the first “infinite” tree and is usually called

ω

THE FIRST INFINITE TREE

The tree



has the property

- It can be embedded in any tree which is not purely unary

It is the first “infinite” tree and is usually called

ω

It represents the ordertype of natural numbers

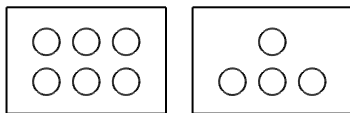
$$\omega = \bullet \bullet \bullet \bullet \dots$$

A MORE COMPLICATED PROCESS

- Situation: A finite number of buckets with balls – bucket 0, bucket 1, bucket 2, . . .
- Step: Take away some balls from the bucket i – and you may put any number of balls in the buckets j where $j < i$
- Termination: No balls left

A MORE COMPLICATED PROCESS

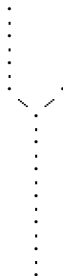
- Situation: A finite number of buckets with balls – bucket 0, bucket 1, bucket 2, ...
- Step: Take away some balls from the bucket i – and you may put any number of balls in the buckets j where $j < i$
- Termination: No balls left



A MORE COMPLICATED PROCESS

Measuring the state above

- Pairs of finite numbers ordered lexicographically: $(6, 4)$
- Ordinals: $\omega \cdot 4 + 6$
- Trees with unary and binary branchings:



A MORE COMPLICATED PROCESS

A finite number of buckets

- Lexicographical ordering of finite sequences of natural numbers – order first with respect to the length of the sequences, then look at the rightmost place where the two sequences differ
- Ordinals less than ω^ω
- Trees generated by

$$x \mapsto \begin{array}{c} x \\ | \\ \cdot \end{array}$$

$$x \mapsto \begin{array}{c} x \quad \cdot \\ \diagdown \quad \diagup \\ \cdot \end{array}$$

A MORE COMPLICATED PROCESS

Termination using binary trees

- Given the history of such a process
- To each state assign a binary tree
- We get a sequence of binary trees B_0, B_1, B_2, \dots
- If the sequence is infinite, there must be $i < j$ such that B_i can be embedded into B_j
- This is impossible
- The sequence must be finite and the process must terminate

A MORE COMPLICATED PROCESS

Termination using binary trees

- Given the history of such a process
- To each state assign a binary tree
- We get a sequence of binary trees B_0, B_1, B_2, \dots
- If the sequence is infinite, there must be $i < j$ such that B_i can be embedded into B_j
- This is impossible
- The sequence must be finite and the process must terminate

The first tree which can be embedded in all trees which are not of the form above is



It represents the ordinal ω^ω

TREES AND EMBEDDINGS

Our trees are finite with root at the bottom and the nodes are ordered from left to right.

Tree S can be embedded in tree T if there are injections

- From nodes of S into nodes of T

TREES AND EMBEDDINGS

Our trees are finite with root at the bottom and the nodes are ordered from left to right.

Tree S can be embedded in tree T if there are injections

- From nodes of S into nodes of T
- From edges of S into paths of T respecting

TREES AND EMBEDDINGS

Our trees are finite with root at the bottom and the nodes are ordered from left to right.

Tree S can be embedded in tree T if there are injections

- From nodes of S into nodes of T
- From edges of S into paths of T respecting
 - the injections of nodes

TREES AND EMBEDDINGS

Our trees are finite with root at the bottom and the nodes are ordered from left to right.

Tree S can be embedded in tree T if there are injections

- From nodes of S into nodes of T
- From edges of S into paths of T respecting
 - the injections of nodes
 - the ordering of the nodes in the trees

TREES AND EMBEDDINGS

Our trees are finite with root at the bottom and the nodes are ordered from left to right.

Tree S can be embedded in tree T if there are injections

- From nodes of S into nodes of T
- From edges of S into paths of T respecting
 - the injections of nodes
 - the ordering of the nodes in the trees
 - none of the paths meet at an interior point

TREES AND EMBEDDINGS

Our trees are finite with root at the bottom and the nodes are ordered from left to right.

Tree S can be embedded in tree T if there are injections

- From nodes of S into nodes of T
- From edges of S into paths of T respecting
 - the injections of nodes
 - the ordering of the nodes in the trees
 - none of the paths meet at an interior point

TREES AND EMBEDDINGS

Our trees are finite with root at the bottom and the nodes are ordered from left to right.

Tree S can be embedded in tree T if there are injections

- From nodes of S into nodes of T
- From edges of S into paths of T respecting
 - the injections of nodes
 - the ordering of the nodes in the trees
 - none of the paths meet at an interior point

The last condition gives what is called a **topological** embedding – the branchings in the embedded tree is not larger than the tree it embeds into. When we talk about embeddings we mean topological embeddings.

HIGMANS LEMMA

Theorem (Higman)

Given an infinite sequence of binary trees B_0, B_1, B_2, \dots . Then there are $i < j$ such that B_i embeds (topologically) into B_j .

We have here formulated the theorem in terms of binary trees – it is common to formulate it in terms of finite sequences and an embedding relation between sequences.

HIGMANS LEMMA

Sketch of proof

- Bad tree: A tree starting an infinite sequence contradicting Higman's lemma

HIGMANS LEMMA

Sketch of proof

- Bad tree: A tree starting an infinite sequence contradicting Higman's lemma
- Good tree: Not bad tree

HIGMANS LEMMA

Sketch of proof

- Bad tree: A tree starting an infinite sequence contradicting Higman's lemma
- Good tree: Not bad tree
- Minimal bad sequence: A sequence of bad trees such that for each tree its immediate subtrees are good

HIGMANS LEMMA

Sketch of proof

- Bad tree: A tree starting an infinite sequence contradicting Higman's lemma
- Good tree: Not bad tree
- Minimal bad sequence: A sequence of bad trees such that for each tree its immediate subtrees are good
- There is no minimal bad sequence.

HIGMANS LEMMA

Sketch of proof

- Bad tree: A tree starting an infinite sequence contradicting Higman's lemma
- Good tree: Not bad tree
- Minimal bad sequence: A sequence of bad trees such that for each tree its immediate subtrees are good
- There is no minimal bad sequence.
- Given a bad tree. Then there is a minimal bad sequence.

HIGMANS LEMMA

Incompleteness results – 1

The smallest tree which can be embedded into any non-binary tree is



It represents Gentzens famous ordinal ϵ_0 :

- Iterations of lexicographical ordering
- The ordinal

$\omega^{\omega^{\omega^{\dots}}}$

HIGMANS LEMMA

Incompleteness results – 2

The Gödel sentence for a system S is the sentence expressing the S is consistent. It is the same for all systems (satisfying some simple conditions) – and give very little information about the system itself.

HIGMANS LEMMA

Incompleteness results – 2

The Gödel sentence for a system S is the sentence expressing the S is consistent. It is the same for all systems (satisfying some simple conditions) – and give very little information about the system itself. The most common formal system is Peano arithmetic \mathcal{PA} . It consists of predicate logic, some simple arithmetic and induction over arbitrary formulas.

HIGMANS LEMMA

Incompleteness results – 2

The Gödel sentence for a system S is the sentence expressing the S is consistent. It is the same for all systems (satisfying some simple conditions) – and give very little information about the system itself. The most common formal system is Peano arithmetic \mathcal{PA} . It consists of predicate logic, some simple arithmetic and induction over arbitrary formulas. Gentzen showed that the Gödel sentence for \mathcal{PA} is equivalent to transfinite induction up to ϵ_0 .

HIGMANS LEMMA

Incompleteness results – 2

The Gödel sentence for a system S is the sentence expressing the S is consistent. It is the same for all systems (satisfying some simple conditions) – and give very little information about the system itself. The most common formal system is Peano arithmetic \mathcal{PA} . It consists of predicate logic, some simple arithmetic and induction over arbitrary formulas. Gentzen showed that the Gödel sentence for \mathcal{PA} is equivalent to transfinite induction up to ϵ_0 . We can formulate it as follows:

In \mathcal{PA} we can perform induction up to any iteration of the lexicographical ordering – but not more than that. This is the limit of what we can do.

HIGMANS LEMMA

Incompleteness results – 2

The Gödel sentence for a system S is the sentence expressing the S is consistent. It is the same for all systems (satisfying some simple conditions) – and give very little information about the system itself. The most common formal system is Peano arithmetic \mathcal{PA} . It consists of predicate logic, some simple arithmetic and induction over arbitrary formulas. Gentzen showed that the Gödel sentence for \mathcal{PA} is equivalent to transfinite induction up to ϵ_0 . We can formulate it as follows:

In \mathcal{PA} we can perform induction up to any iteration of the lexicographical ordering – but not more than that. This is the limit of what we can do.

In \mathcal{PA} we can give a proof for each particular binary tree that it is good, but there is no uniform proof in \mathcal{PA} that all binary trees are good.

THE PROCESSES OF ARITHMETIC

- Normalization: This is the process we use to evaluate a program in a functional language given an input. The functional language is often taken to be Gödel's T and is an important fragment of most functional languages.

THE PROCESSES OF ARITHMETIC

- Normalization: This is the process we use to evaluate a program in a functional language given an input. The functional language is often taken to be Gödel's T and is an important fragment of most functional languages.
- Cut elimination: This is the process where in a derivation we eliminate auxiliary lemmas and end up with a direct proof.

THE PROCESSES OF ARITHMETIC

- Normalization: This is the process we use to evaluate a program in a functional language given an input. The functional language is often taken to be Gödel's T and is an important fragment of most functional languages.
- Cut elimination: This is the process where in a derivation we eliminate auxiliary lemmas and end up with a direct proof.

THE PROCESSES OF ARITHMETIC

- Normalization: This is the process we use to evaluate a program in a functional language given an input. The functional language is often taken to be Gödel's T and is an important fragment of most functional languages.
- Cut elimination: This is the process where in a derivation we eliminate auxiliary lemmas and end up with a direct proof.

There is an extensive literature in proof theory tying such processes up to the ordinal ϵ_0 and hence to Higman's lemma.

KRUSKALS THEOREM

The same formulation as in Higman's lemma but with no restriction to binary trees.

Theorem (Kruskal)

Given an infinite sequence of finite trees T_0, T_1, T_2, \dots . Then there are $i < j$ such that T_i embeds (topologically) into T_j .

Using Kruskal's theorem we can control much more complicated processes. At the moment there is much activity within proof theory of Kruskal's theorem and trying to characterize the kind of processes it can control. The limit corresponds to an ordinal known as the small Veblen ordinal and given by work by Oswald Veblen 100 years ago.

KRUSKALS GAP THEOREM

There is an obvious generalization of Kruskals theorem to finite trees with labels at the nodes. The labels are then from a wellordered set Λ given in advance. We just have to make the embedding respect the labels. This does not really give anything new, but can be nice with respect to applications. Often the trees come from terms – and terms can be considered as labeled trees.

KRUSKALS GAP THEOREM

There is an obvious generalization of Kruskals theorem to finite trees with labels at the nodes. The labels are then from a wellordered set Λ given in advance. We just have to make the embedding respect the labels. This does not really give anything new, but can be nice with respect to applications. Often the trees come from terms – and terms can be considered as labeled trees.

Kruskals gap theorem goes further. Here we give extra conditions to the embeddings. As before we have injections of nodes into nodes and edges into paths. We now require that all internal nodes in the paths have labels \geq than the label at the end of the path.

KRUSKALS GAP THEOREM

Proof theory

The proof theoretic variant of Kruskals gap theorem (from 1990) was developed by Gaisi Takeuti 1955-1965. Using just two labels we can control inductive definition (as we use in Borel sets). With finite number of labels we can get iterated inductive definitions. Using some small infinite ordinal as label we get so called Π_1^1 - CA and much more with larger ordinals as labels.

GRAPH MINOR THEOREM

Theorem (Robertson, Seymour)

Given an infinite sequence of finite graphs G_0, G_1, G_2, \dots there are $i < j$ such that G_i is embedded into G_j .

This fits well into the development here but for a proof theorist we can say that it goes a long way beyond the ordinary Kruskals theorem – but you know more about it here than I do.