universität innsbruck

Specialisation Seminar

Abstraction Interpretations

Georg Moser

cbr.uibk.ac.at

A Textbook Example

Example

```
let rec fold_left f acc = function
    [] -> acc
    | x::xs -> fold_left f (f acc x) xs
;;
let rev ls = fold_left (fun xs x -> x::xs) [] ls
;;
```

A Textbook Example

Example

```
let rec fold_left f acc = function
    [] -> acc
    | x::xs -> fold_left f (f acc x) xs
;;
let rev ls = fold_left (fun xs x -> x::xs) [] ls
;;
```

Question

What is the runtime complexity of rev?

A Textbook Example

Example

```
let rec fold_left f acc = function
    [] -> acc
    | x::xs -> fold_left f (f acc x) xs
;;
let rev ls = fold_left (fun xs x -> x::xs) [] ls
;;
```

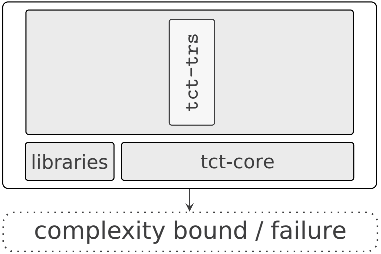
Question

What is the runtime complexity of rev?

TcT says

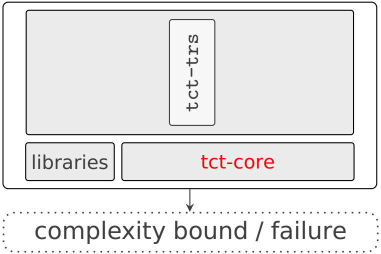
$O(n)$

Tyrolean Complexity Tool (`tct-trs`)



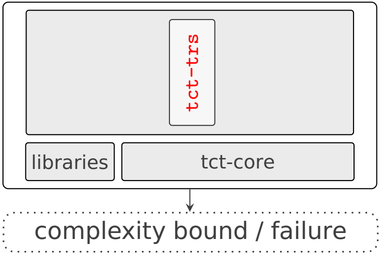
- fully automated complexity analysis tool for TRSs
- recipient of a Gödel medal in the 1st FLOC Olympic games
- modular complexity analysis framework
- partly certified by `CeTA`

Tyrolean Complexity Tool (`tct-trs`)



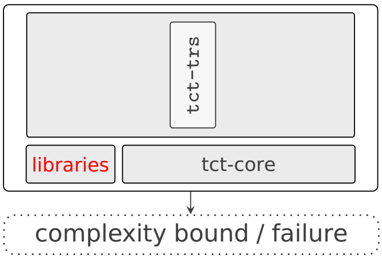
- fully automated complexity analysis tool for TRSs
- recipient of a Gödel medal in the 1st FLOC Olympic games
- modular complexity analysis framework
- partly certified by `CeTA`

Tyrolean Complexity Tool (`tct-trs`)



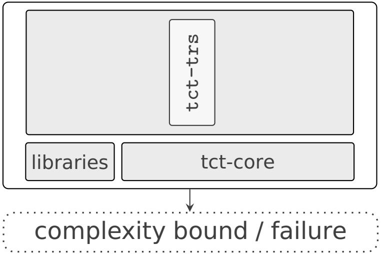
- fully automated complexity analysis tool for TRSs
- recipient of a Gödel medal in the 1st FLOC Olympic games
- modular complexity analysis framework
- partly certified by CeTA

Tyrolean Complexity Tool (tct-trs)



- fully automated complexity analysis tool for TRSs
- recipient of a Gödel medal in the 1st FLOC Olympic games
- modular complexity analysis framework
- partly certified by CeTA

Tyrolean Complexity Tool (`tct-trs`)



- fully automated complexity analysis tool for TRSs
- recipient of a Gödel medal in the 1st FLC
- mo s fra
- par

M. Avanzini, C. Sternagel, and R. Thiemann.
Certification of complexity proofs using CeTA.
In *Proc. 26th RTA*, volume 36 of *LIPIcs*, pages 23–39, 20

M. Avanzini and GM.
A combination framework for complexity.
IC, 248:22–55, 2016.

M. Avanzini, GM, and M. Schaper.
Tct: Tyrolean complexity tool.
In *Proc. 22nd TACAS*, volume 9636 of *LNCS*, pages 407–423, 2016.

Does Testing Suffice?

Problems

- only samples the set of possible behaviors
- unlike physical systems, software systems are discontinuous
- there is no sound basis for extrapolating from tested to untested cases
- need to consider all (= infinitely many cases) . . . is this possible?
- it's possible with symbolic techniques
- for example Abstract Interpretations
- more on that in the seminar

Importance of Static Analysis

In the Beginning

- static analysis initially used for optimising compilers
- program transformations preserve the behavior
- still an important application for static analysis.

Importance of Static Analysis

In the Beginning

- static analysis initially used for optimising compilers
- program transformations preserve the behavior
- still an important application for static analysis.

Today

- bug finding
- program understanding
- verification
- resource analysis

The Phases of GCC

- parsing
- tree optimisation
- RTL generation
- sibling call optimisation
- jump optimisation
- register scan
- jump threading
- common subexpression elimination
- loop optimisations
- jump bypassing
- data flow analysis
- instruction combination
- if-conversion
- register movement
- instruction scheduling
- register allocation
- basic block reordering
- delayed branch scheduling
- branch shortening
- assembly output
- debugging output

The Phases of GCC

- parsing
- tree optimisation
- RTL generation
- sibling call optimisation
- jump optimisation
- register scan
- jump threading
- common subexpression elimination
- loop optimisations
- jump bypassing
- data flow analysis
- instruction combination
- if-conversion
- register movement
- instruction scheduling
- register allocation
- basic block reordering
- delayed branch scheduling
- branch shortening
- assembly output
- debugging output

60% of the compiliation time spent in static analysis

Availabe Expression Analysis

```
i := 0;
while (i <= n) {
  j := 0;
  while (j <= m) {
    A[i*(m+1)+j] := B[i*(m+1)+j] + C[i*(m+1)+j] ;
    j := j+1; }
  i := i+1; }
```

Availabe Expression Analysis

```
i := 0;
while (i <= n) {
  j := 0;
  while (j <= m) {
    A[i*(m+1)+j] := B[i*(m+1)+j] + C[i*(m+1)+j] ;
    j := j+1; }
  i := i+1; }
```

Common Subexpression Elimination: Introduction of Temp Vars

```
i := 0;
while (i <= n) {
  j := 0;
  while (j <= m) {
    temp := i*(m+1)+j;
    A[temp] := B[temp] + C[temp] ;
    j := j+1; }
  i := i+1; }
```

Theoretical Results

Theorem (Rice's Theorem)

Interesting Program Properties are undecidable.

Proof.

See FLAT ∎

Theoretical Results

Theorem (Rice's Theorem)

Interesting Program Properties are undecidable.

Proof.

See FLAT ∎

Undecidabilty

- static analysis is (very) undecidable, that is necessarily unsound or incomplete or partial
- engineering challenge is to give practical useful answers anyway

Labelled Programs

Definitions (syntactic categories)

$a \in \mathbf{AExp}$	arithmetic expressions	$x, y \in \mathbf{Var}$	variables
$b \in \mathbf{BExp}$	Boolean expressions	$n \in \mathbf{Num}$	numerals
$S \in \mathbf{STmt}$	statments	$\ell \in \mathbf{Lab}$	labels

Labelled Programs

Definitions (syntactic categories)

$a \in \mathbf{AExp}$	arithmetic expressions	$x, y \in \mathbf{Var}$	variables
$b \in \mathbf{BExp}$	Boolean expressions	$n \in \mathbf{Num}$	numerals
$S \in \mathbf{STmt}$	statments	$\ell \in \mathbf{Lab}$	labels

Definitions (operators)

$op_a \in \mathbf{Op}_a$	arithmetic operators
$op_b \in \mathbf{Op}_b$	Boolean operators
$op_r \in \mathbf{Op}_r$	relational operators

Abstract Syntax

Definition (syntax of WHILE)

$$a ::= x \mid n \mid a_1\ op_a\ b_2$$

$$b ::= \textbf{true} \mid \textbf{false} \mid \textbf{not}\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2$$

$$S ::= [x := a]^\ell \mid [\textbf{skip}]^\ell \mid S_1; S_2 \mid \textbf{if}\ [b]^\ell\ \textbf{then}\ S_1 \textbf{else}\ S_2 \mid \textbf{while}\ [b]^\ell\ \textbf{do}\ S$$

Abstract Syntax

Definition (syntax of WHILE)

$$a ::= x \mid n \mid a_1\ op_a\ b_2$$

$$b ::= \textbf{true} \mid \textbf{false} \mid \textbf{not}\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2$$

$$S ::= [x := a]^\ell \mid [\textbf{skip}]^\ell \mid S_1; S_2 \mid \textbf{if}\ [b]^\ell\ \textbf{then}\ S_1 \textbf{else}\ S_2 \mid \textbf{while}\ [b]^\ell\ \textbf{do}\ S$$

Konvention

- labelled items are referred to as elementary blocks
- we shall use paranteses to disambiguate the syntax
- for statements we write $\{,\}$ or **begin**, **end**
- for other categories we use round brackets

Reaching Definition Analysis

Definition

An assignment (aka definition) of the from $[x := a]^\ell$ may reach a certain program point if there is an execution of the program where x was last assigned a value at ℓ when the program point is reached

Reaching Definition Analysis

Definition

An assignment (aka definition) of the from $[x := a]^\ell$ may reach a certain program point if there is an execution of the program where x was last assigned a value at ℓ when the program point is reached

Definitions

- $(y, 1)$ reaches the entry to 2, if the assignment $[y := x]^1$ reaches the entry of the elementary block $[z := 1]^2$
- we also say that $(x, ?)$ reaches 2; here ? stands for an uninitialised value

full information about reaching definitions is given by the pair $\mathrm{RD} = (\mathrm{RD}_{\mathrm{entry}}, \mathrm{RD}_{\mathrm{exit}})$ for each program location

Data Flow Analysis

The Equational Approach

an analysis (like reaching definitions) can be specified by extracting a number of equations from the program

Data Flow Analysis

The Equational Approach

an analysis (like reaching definitions) can be specified by extracting a number of equations from the program

Equation System

there are two classes of equations:

- relate exit information of a block to entry information of the same block
- relate entry information of a block to the exit information of the preceeding block (wrt the CFG)
- the CFG may be immediately obvious (as in the WHILE language) or may be obtained by a control flow analysis