Verification of Functional Programs
Analysis of Algorithms

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2026-1

Introduction

Preliminaries

Textbook

Tobias Nipkow ([2025] 2026). Functional Data Structures and Algorithms.

Convention

The numbers and page numbers assigned to chapters, examples, exercises, figures, quotes, sections and theorems on these slides correspond to the numbers assigned in the textbook.

Analysis of Algorithms

Description

The **computational complexity** of an algorithm/problem is the amount of resources (e.g. time, space, energy) required to execute/solve it.

Description

The **analysis of algorithms**—term coined by Donald Knuth—is the study of the computational complexity of algorithms. A classical book is (Cormen, Leiserson, Rivest and Stein [1990] 2022).

Description

The **computational complexity theory** is the study of the computacional complexity of problems. A classical book is (Garey and Johnson 1979).

Analysis of Algorithms

Relation between analysis of algorithms and computational complexity theory

> *"The [computational] complexity of an algorithm is always an upper bound on the [computational] complexity of the problem solved by this algorithm."*
> *(Wikipedia, Computational complexity, 2026-03-16).*

Analysis of Algorithms

Two abstractions

For the analysis of algorithms we required two abstractions:

Analysis of Algorithms

Two abstractions

For the analysis of algorithms we required two abstractions:

(i) Where do the algorithms run? In a theoretical computer, i.e., we are interested in machine-independent algorithms.

Analysis of Algorithms

Two abstractions

For the analysis of algorithms we required two abstractions:

(i) Where do the algorithms run? In a theoretical computer, i.e., we are interested in machine-independent algorithms.

(ii) Which complexity are we interested? We are interested in **asymptotic complexity**, i.e., we are interested in the behaviour of the algorithm for large values of the input.

Analysis of Algorithms

The running time function

If the running time of an algorithm depends of the input then it usually means it depends of the size of the input.

We shall use a function

$$T(n) : \mathbb{N} \to \mathbb{R}^{\geq 0}$$

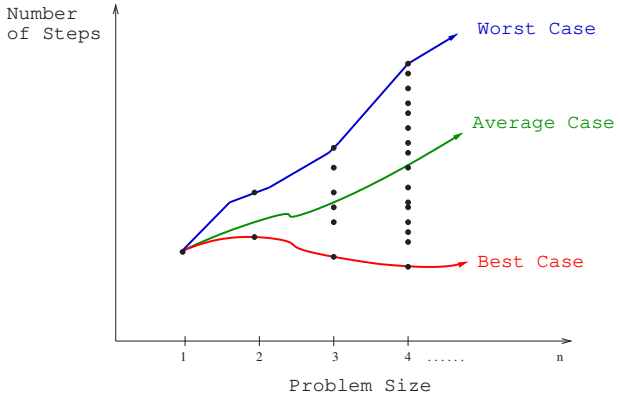
which will denote the running time of an algorithm on inputs of size n.

Asymptotic Notations

Complexity Functions

Complexity functions

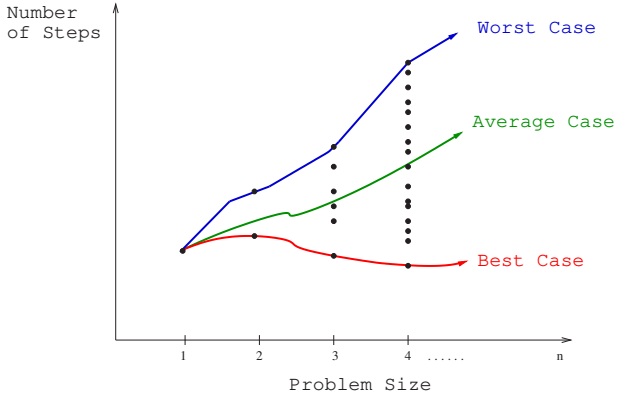
Given an input of size n we can think in three complexity functions: **best-case complexity**, **worst-case complexity** and **average-case** complexity.



Complexity Functions

Complexity functions

Given an input of size n we can think in three complexity functions: **best-case complexity**, **worst-case complexity** and **average-case** complexity.[†]



[†]Figure from Skiena ([1997] 2020, Fig. 2.1).

Asymptotic Notation: Big O

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big** O **of** $g(n)$, denoted $O(g(n))$, by

$$O(g(n)) \coloneqq \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c \in \mathbb{R}^{+} \text{ and } n_0 \in \mathbb{Z}^{+} \text{ such that } f(n) \leq cg(n) \text{ for all } n \geq n_0 \,\}.$$

Asymptotic Notation: Big O

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big O of $g(n)$**, denoted $O(g(n))$, by

$$O(g(n)) \coloneqq \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c \in \mathbb{R}^{+} \text{ and } n_0 \in \mathbb{Z}^{+} \text{ such that } f(n) \leq cg(n) \text{ for all } n \geq n_0 \,\}.$$

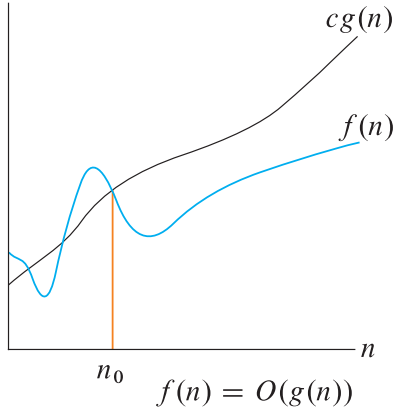
Notation

That $f(n) \in O(g(n))$ is sometimes denoted by $f(n) = O(g(n))$.

Asymptotic Notation: Big O

Remark

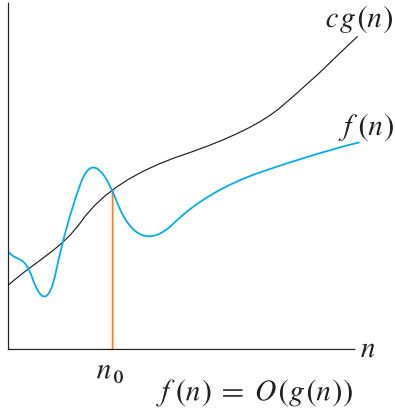
If $f(n) \in O(g(n))$ then function $g(n)$ is an upper bound on the growth rate of the function $f(n)$.



Asymptotic Notation: Big O

Remark

If $f(n) \in O(g(n))$ then function $g(n)$ is an upper bound on the growth rate of the function $f(n)$.[†]



[†]Figure from Cormen, Leiserson, Rivest and Stein ([1990] 2022, Fig. 3.1a).

Asymptotic Notation: Big Ω

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big Ω of** $g(n)$, denoted $\Omega(g(n))$, by

$$\Omega(g(n)) = \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c \in \mathbb{R}^{+} \text{ and } n_0 \in \mathbb{Z}^{+} \text{ such that } f(n) \geq cg(n) \text{ for all } n \geq n_0 \,\}.$$

Asymptotic Notation: Big Ω

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big Ω of** $g(n)$, denoted $\Omega(g(n))$, by

$$\Omega(g(n)) = \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c \in \mathbb{R}^{+} \text{ and } n_0 \in \mathbb{Z}^{+} \text{ such that } f(n) \geq cg(n) \text{ for all } n \geq n_0 \,\}.$$

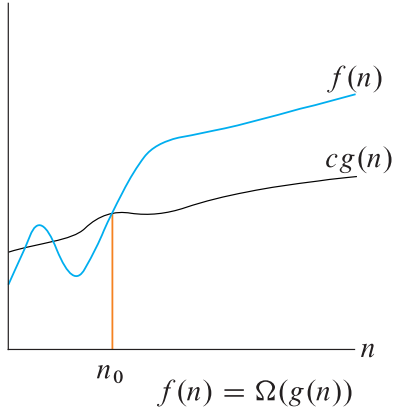
Notation

That $f(n) \in \Omega(g(n))$ is sometimes denoted by $f(n) = \Omega(g(n))$.

Asymptotic Notation: Big Ω

Remark

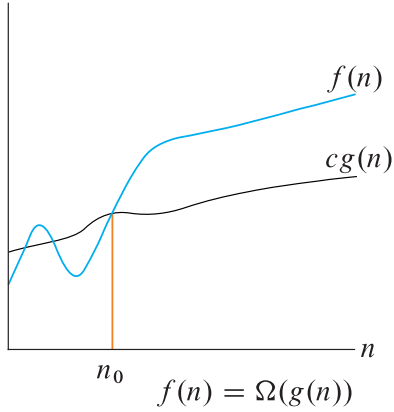
If $f(n) \in \Omega(g(n))$ then function $g(n)$ is a lower bound on the growth rate of the function $f(n)$.



Asymptotic Notation: Big Ω

Remark

If $f(n) \in \Omega(g(n))$ then function $g(n)$ is a lower bound on the growth rate of the function $f(n)$.[†]



[†]Figure from Cormen, Leiserson, Rivest and Stein ([1990] 2022, Fig. 3.1b).

Asymptotic Notation: Big Θ

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big Θ of** $g(n)$, denoted $\Theta(g(n))$, by

$$\Theta(g(n)) = \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c_1, c_2 \in \mathbb{R}^+ \text{ and } n_0 \in \mathbb{Z}^+ \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \,\}.$$

Asymptotic Notation: Big Θ

Definition

Let $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be a function. We define the set of functions **big Θ of** $g(n)$, denoted $\Theta(g(n))$, by

$$\Theta(g(n)) = \{\, f : \mathbb{N} \to \mathbb{R}^{\geq 0} \mid \text{there exist positive constants } c_1, c_2 \in \mathbb{R}^+ \text{ and } n_0 \in \mathbb{Z}^+ \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \,\}.$$

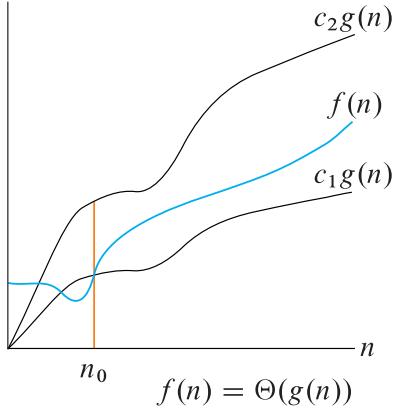
Notation

That $f(n) \in \Theta(g(n))$ is sometimes denoted by $f(n) = \Theta(g(n))$.

Asymptotic Notation: Big Θ

Remark

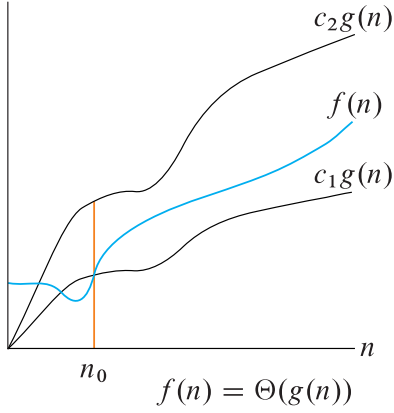
If $f(n) \in \Theta(g(n))$ then function $g(n)$ is a lower bound and an upper bound on the growth rate of the function $f(n)$.



Asymptotic Notation: Big Θ

Remark

If $f(n) \in \Theta(g(n))$ then function $g(n)$ is a lower bound and an upper bound on the growth rate of the function $f(n)$.[†]



[†]Figure from Cormen, Leiserson, Rivest and Stein ([1990] 2022, Fig. 3.1c).

The Tyranny of Growth Rate

The Tyranny of Growth Rate

Growing rates of some functions

Each operation takes one nanosecond (10^9 seconds).

$n/T(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 yrs
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 yrs		

The Tyranny of Growth Rate

Growing rates of some functions

Each operation takes one nanosecond (10^9 seconds).[†]

$n/T(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 yrs
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 yrs		

[†]Table adapted from (Skiena [1997] 2020, Fig. 2.2).

The Tyranny of Growth Rate

Supercomputers

Machines from: `www.top500.org` (last updated: March 2026)
PetaFLOP (PFLOP): 10^{15} floating-point operations per second

Date	Machine	PFLOPs
2026-11	El Capitan	1,809.00
2020-06	Fugaku	415.53
2019-06	Summit	148.60
2018-11	Summit	143.50
2018-06	Summit	122.30
2016-06	Sunway TaihuLight	93.01
2013-06	Tianhe-2	33.86
2012-06	Blue Gene/Q	16.32
2011-06	K computer	8.16

The Tyranny of Growth Rate

Example (3-SAT problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

The Tyranny of Growth Rate

Example (3-SAT problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

A (propositional logic) formula F is in **conjunctive normal form** iff

$$F \text{ has the form } F_1 \wedge \cdots \wedge F_n,$$

where each $F_1, \ldots, F_n$ is a disjunction of literals.

The Tyranny of Growth Rate

Example (3-SAT problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

A (propositional logic) formula F is in **conjunctive normal form** iff

$$F \text{ has the form } F_1 \wedge \cdots \wedge F_n,$$

where each $F_1, \ldots, F_n$ is a disjunction of literals.

3-SAT problem: To determine the satisfiability of a propositional formula in conjunctive normal form where each disjunction of literals is limited to at most three literals.

The Tyranny of Growth Rate

Example (3-SAT problem)

A **literal** is an atomic formula (propositional variable) or the negation of an atomic formula.

A (propositional logic) formula F is in **conjunctive normal form** iff

$$F \text{ has the form } F_1 \wedge \cdots \wedge F_n,$$

where each $F_1, \ldots, F_n$ is a disjunction of literals.

3-SAT problem: To determine the satisfiability of a propositional formula in conjunctive normal form where each disjunction of literals is limited to at most three literals.

The problem was proposed in Karp's 21 NP-complete problems (Karp 1972).

The Tyranny of Growth Rate

Example (improvements on the time complexity of 3-SAT deterministic algorithmic)

$O(1.32793^n)$	Liu (2018)
$O(1.3303^n)$	Makino, Tamaki and Yamamoto (2011) and Makino, Tamaki and Yamamoto (2013)
$O(1.3334^n)$	Moser and Scheder (2011)
$O(1.439^n)$	Kutzkov and Scheder (2010)
$O(1.465^n)$	Scheder (2008)
$O(1.473^n)$	Brueggemann and Kern (2004)
$O(1.481^n)$	Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan and Schöning (2002)

(continued on next slide)

The Tyranny of Growth Rate

Example (improvements on the time complexity of 3-SAT deterministic algorithmic[†])

$O(1.32793^n)$	Liu (2018)
$O(1.3303^n)$	Makino, Tamaki and Yamamoto (2011) and Makino, Tamaki and Yamamoto (2013)
$O(1.3334^n)$	Moser and Scheder (2011)
$O(1.439^n)$	Kutzkov and Scheder (2010)
$O(1.465^n)$	Scheder (2008)
$O(1.473^n)$	Brueggemann and Kern (2004)
$O(1.481^n)$	Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan and Schöning (2002)

(continued on next slide)

[†]Main sources: Hertli (2011) and Hertli (2015). Last updated: March 2026.

The Tyranny of Growth Rate

Example (continuation)

$O(1.497^n)$	Schiermeyer (1996)
$O(1.505^n)$	Kullmann (1999)
$O(1.6181^n)$	Monien and Speckenmeyer (1979) and Monien and Speckenmeyer (1985)
$O(2^n)$	Brute-force search

The Tyranny of Growth Rate

Example (3-SAT simulation)

Running 3-SAT times on different supercomputers using the faster deterministic algorithm, i.e. $T(n) = 1.32793^n$.

Date	Machine	PFLOPs	$n = 150$	$n = 200$	$n = 400$
2025-11	El Capitan	1,809.00	1.7 sec	27.6 days	3.2×10^{23} yrs
2020-06	Fugaku	415.53	7.2 sec	120.2 days	1.4×10^{24} yrs
2019-06	Summit	148.60	20.1 sec	336.1 days	4.0×10^{24} yrs
2018-11	Summit	143.50	20.8 sec	348.1 days	4.1×10^{24} yrs
2018-06	Summit	122.30	24.5 sec	1.1 yrs	4.8×10^{24} yrs
2016-06	Sunway TaihuLight	93.01	32.2 sec	1.5 yrs	6.4×10^{24} yrs
2013-06	Tianhe-2	33.86	1.5 min	4.1 yrs	1.7×10^{25} yrs
2012-06	Blue Gene/Q	16.32	3.1 min	8.4 yrs	3.6×10^{25} yrs
2011-06	K computer	8.16	6.1 min	16.8 yrs	7.3×10^{25} yrs

The Tyranny of Growth Rate

Example (3-SAT simulation)

Running 3-SAT times for different deterministic algorithms using the faster supercomputer, i.e. 1,809.00 PFLOPs.

$T(n)$	$n = 150$	$n = 200$	$n = 400$
1.32793^n	1.7 sec	27.6 days	3.2×10^{23} yrs
1.3303^n	2.1 sec	39.4 days	6.7×10^{23} yrs
1.3334^n	3.1 sec	62.8 days	1.7×10^{24} yrs
1.439^n	3.3 days	7.2×10^5 yrs	2.9×10^{37} yrs
1.465^n	48.0 days	2.6×10^7 yrs	3.8×10^{40} yrs
2^n	2.5×10^{19} yrs	2.8×10^{34} yrs	4.5×10^{94} yrs

Running Time Functions in `Isabelle/HOL`

Running Time Functions in Isabelle/HOL

Description

Given a function $f : \sigma_1 \to \cdots \to \sigma_n \to \tau$ we define a **running time function** $T_f : \sigma_1 \to \cdots \to \sigma_n \to \mathrm{Nat}$ by

$$\mathcal{E}[\![f\ p_1 \ldots p_n = e]\!] = (T_f\ p_1 \ldots p_n = \mathcal{T}[\![e]\!] + 1),$$
$$\mathcal{T}[\![f\ e_1 \ldots e_n]\!] = \mathcal{T}[\![e_1]\!] + \cdots + T[\![e_n]\!] + T_f\ e_1 \ldots e_n,$$

where *"$\mathcal{E}[\![\cdot]\!]$ translates defining equations for f to defining equations for T_f and $\mathcal{T}[\![\cdot]\!]$ translates expressions that compute some value to expressions that computes the number of function calls"* (p. 6).

Running Time Functions in Isabelle/HOL

Convention

> *"Conceptually we define* $T_f \cdots = 0$ *if* f *is a variable, constructor function or predefined function on bool or numbers. That is, we count only user-defined function calls." (p. 7)*

Running Time Functions in Isabelle/HOL

Special case

$$\mathcal{T}[\![\text{if } b \text{ then } e_1 \text{ else } e_2]\!] = \mathcal{T}[\![b]\!] + (\text{if } b \text{ then } \mathcal{T}[\![e_1]\!] \text{ else } \mathcal{T}[\![e_2]\!])$$

Running Time Functions in Isabelle/HOL

Example

We calculate the running time function T_{app} for the function app defined by

$$\begin{aligned}
&\text{app} : \text{List}\ \tau \to \text{List}\ \tau \to \text{List}\ \tau \\
&\text{app}\ [\,]\ \quad\quad\ ys = ys \\
&\text{app}\ (x \# xs)\ ys = x \# (\text{app}\ xs\ ys)
\end{aligned}$$

(continued on next slide)

Running Time Functions in Isabelle/HOL

Example (continuation)

$$\begin{aligned}
&\mathcal{E}[\![\,\text{app}\ [\,]\ ys = ys\,]\!] \\
&\quad = (T_{\text{app}}\ [\,]\ ys = \mathcal{T}[\![\,ys\,]\!] + 1) \\
&\quad = (T_{\text{app}}\ [\,]\ ys = 1) \qquad \text{(because } \mathcal{T}[\![\,ys\,]\!] = 0\text{)}
\end{aligned}$$

$$\begin{aligned}
&\mathcal{E}[\![\,\text{app}\ (x\ \#\ xs)\ ys = x\ \#\ (\text{app}\ xs\ ys)\,]\!] \\
&\quad = (T_{\text{app}}\ (x\ \#\ xs)\ ys = \mathcal{T}[\![\,x\ \#\ (\text{app}\ xs\ ys)\,]\!] + 1)
\end{aligned}$$

$$\begin{aligned}
&\mathcal{T}[\![\,x\ \#\ (\text{app}\ xs\ ys)\,]\!] \\
&\quad = \mathcal{T}[\![\,x\,]\!] + \mathcal{T}[\![\,\text{app}\ xs\ ys\,]\!] + T_{\text{cons}}\ x\ (\text{app}\ xs\ ys) \\
&\quad = \mathcal{T}[\![\,\text{app}\ xs\ ys\,]\!] \qquad \text{(because } \mathcal{T}[\![\,x\,]\!] = T_{\text{cons}}\ x\ (\text{app}\ xs\ ys) = 0\text{)} \\
&\quad = \mathcal{T}[\![\,xs\,]\!] + \mathcal{T}[\![\,ys\,]\!] + T_{\text{app}}\ xs\ ys \\
&\quad = T_{\text{app}}\ xs\ ys \qquad \text{(because } \mathcal{T}[\![\,xs\,]\!] = \mathcal{T}[\![\,ys\,]\!] = 0\text{)}
\end{aligned}$$

(continued on next slide)

Running Time Functions in Isabelle/HOL

Example (continuation)

From the previous equations we have

$$T_{\text{app}} : \text{List } \tau \to \text{List } \tau \to \text{Nat}$$
$$T_{\text{app}}\ [\,]\ \ ys = 1$$
$$T_{\text{app}}\ (x \# xs)\ ys = T_{\text{app}}\ xs\ ys + 1$$

Running Time Functions in Isabelle/HOL

Example

We calculate the running time function T_{insort1} for the function insort1 defined by

$$\begin{aligned}
&\text{insort1} : (\text{LinOrder } \tau) \Rightarrow \tau \to \text{List } \tau \to \text{List } \tau \\
&\text{insort1 } x\ [\,] &&= [x] \\
&\text{insort1 } x\ (y \# ys) &&= \text{if } x \leq y \text{ then } x \# (y \# ys) \text{ else } y \# (\text{insort1 } x\ ys)
\end{aligned}$$

(continued on next slide)

Running Time Functions in Isabelle/HOL

Example (continuation)

$$\mathcal{E}[\![\,\text{insort1}\ x\ [\,] = [x]\,]\!]$$
$$= (T_{\text{insort1}}\ x\ [\,] = \mathcal{T}[\![\,[x]\,]\!] + 1)$$
$$= (T_{\text{insort1}}\ x\ [\,] = 1) \qquad \text{(because } \mathcal{T}[\![\,[x]\,]\!] = 0\text{)}$$

$$\mathcal{E}[\![\,\text{insort1}\ x\ (y\ \#\ ys) = \text{if } x \leq y \text{ then } x\ \#\ (y\ \#\ ys) \text{ else } y\ \#\ (\text{insort1}\ x\ ys)\,]\!]$$
$$= (T_{\text{insort1}}\ x\ (y\ \#\ ys) = \mathcal{T}[\![\,\text{if } x \leq y \text{ then } x\ \#\ (y\ \#\ ys) \text{ else } y\ \#\ (\text{insort1}\ x\ ys)\,]\!] + 1)$$

$$\mathcal{T}[\![\,y\ \#\ (\text{insort1}\ x\ ys)\,]\!]$$
$$= \mathcal{T}[\![\,y\,]\!] + \mathcal{T}[\![\,\text{insort1}\ x\ ys\,]\!] + T_{\text{cons}}\ y\ (\text{insort1}\ x\ ys)$$
$$= \mathcal{T}[\![\,\text{insort1}\ x\ ys\,]\!] \qquad \text{(because } \mathcal{T}[\![\,y\,]\!] = T_{\text{cons}}\ y\ (\text{insort1}\ x\ ys) = 0\text{)}$$
$$= \mathcal{T}[\![\,x\,]\!] + \mathcal{T}[\![\,ys\,]\!] + T_{\text{insort1}}\ x\ ys$$
$$= T_{\text{insort1}}\ x\ ys \qquad \text{(because } \mathcal{T}[\![\,x\,]\!] = \mathcal{T}[\![\,ys\,]\!] = 0\text{)}$$

(continued on next slide)

Running Time Functions in Isabelle/HOL

Example (continuation)

$$\begin{aligned}
&\mathcal{T}[\![\,\text{if } x \le y \text{ then } x \# (y \# ys) \text{ else } y \# (\text{insort1 } x\ ys)\,]\!] \\
&= \mathcal{T}[\![\,x \le y\,]\!] + \text{if } x \le y \text{ then } \mathcal{T}[\![\,x \# (y \# ys)\,]\!] \text{ else } \mathcal{T}[\![\,y \# (\text{insort1 } x\ ys)\,]\!] \\
&= \text{if } x \le y \text{ then } 0 \text{ else } \mathcal{T}[\![\,y \# (\text{insort1 } x\ ys)\,]\!] \text{ (because } \mathcal{T}[\![\,x \le y\,]\!] = \mathcal{T}[\![\,x \# (y \# ys)\,]\!] = 0) \\
&= \text{if } x \le y \text{ then } 0 \text{ else } T_{\text{insort1}}\ x\ ys
\end{aligned}$$

(continued on next slide)

Running Time Functions in Isabelle/HOL

Example (continuation)

From the previous equations we have

$$
\begin{aligned}
&T_{\text{insort1}} : (\text{LinOrder } \tau) \Rightarrow \tau \to \text{List } \tau \to \text{Nat} \\
&T_{\text{insort1}}\ x\ [\,] &&= 1 \\
&T_{\text{insort1}}\ x\ (y \# ys) &&= (\text{if } x \leq y \text{ then } 0 \text{ else } T_{\text{insort1}}\ x\ ys) + 1
\end{aligned}
$$

References

References

Tobias Brueggemann and Walter Kern (2004). An Improved Deterministic Local Search Algorithm for 3-SAT. Theoretical Computer Science 329.1–3, pp. 303–313. DOI: 10.1016/j.tcs.2004.08.002 (cit. on pp. 33, 34).

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein [1990] (2022). Introduction to Algorithms. 4th ed. MIT Press (cit. on pp. 4, 16, 20, 24).

Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon Kleinberg, Christos Papadimitriou, Prabhakar Raghavan and Uwe Schöning (2002). A Deterministic $(2 - 2/(k + 1))^n$ Algorithm for k-SAT Based on Local Search. Theoretical Computer Science 289.1, pp. 69–83. DOI: 10.1016/S0304-3975(01)00174-8 (cit. on pp. 33, 34).

Michael R. Garey and David S. Johnson (1979). Computers and Intractability. A Guide to the Theory of NP-completeness. W. H. Freeman and Company (cit. on p. 4).

Timon Hertli (2011). 3-SAT Faster and Simpler - Unique-SAT Bounds for PPSZ Hold in General. In: Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011). IEEE, pp. 277–284. DOI: 10.1109/FOCS.2011.22 (cit. on pp. 33, 34).

Timon Hertli (2015). Improved Exponential Algorithms for SAT and CISP. PhD thesis. ETH Zurich. DOI: 10.3929/ethz-a-010512781 (cit. on pp. 33, 34).

References

Richard M. Karp (1972). Reducibility Among Combinatorial Problems. In: Complexity of Computer Computations. Ed. by Raymond E. Miller and James W. Thatcher. Plenum Press, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9 (cit. on pp. 29–32).

O. Kullmann (1999). New Methods for 3-SAT Decision and Worst-Case Analysis. Theoretical Computer Science 223.1–2, pp. 1–72. DOI: 10.1016/S0304-3975(98)00017-6 (cit. on p. 35).

Konstantin Kutzkov and Dominik Scheder (2010). Using CSP to Improve Deterministic 3-SAT. CoRR abs/1007.1166. URL: https://arxiv.org/abs/1007.1166 (cit. on pp. 33, 34).

Sixue Liu (2018). Chain, Generalization of Covering Code, and Deterministic Algorithm for k-SAT. In: 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018). Ed. by Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx and Donald Sannella. Vol. 107. Leibniz International Proceedings in Informatics (LIPIcs), 88:1–88:13. DOI: 10.4230/LIPIcs.ICALP.2018.88 (cit. on pp. 33, 34).

References

Kazuhisa Makino, Suguru Tamaki and Masaki Yamamoto (2011). Derandomizing HSSW Algorithm for 3-SAT. In: Computing and Combinatorics (COCOON 2011). Ed. by Bin Fu and Ding-Zhu Du. Vol. 6842. Lecture Notes in Computer Science. Springer, pp. 1–12. DOI: 10.1007/978-3-642-22685-4_1 (cit. on pp. 33, 34).

Kazuhisa Makino, Suguru Tamaki and Masaki Yamamoto (2013). Derandomizing HSSW Algorithm for 3-SAT. Algorithmica 67.2, pp. 112–124. DOI: 10.1007/s00453-012-9741-4 (cit. on pp. 33, 34).

B. Monien and E. Speckenmeyer (1979). 3-Satisfiability is Testable in $O(1.62^r)$ Steps. Tech. rep. 3/1979. Reihe Theoretische Informatik, Universität Gesamthochschule Paderborn (cit. on p. 35).

B. Monien and E. Speckenmeyer (1985). Solving Satisfiability in less than 2^n Steps. Discrete Applied Mathematics 10.3, pp. 287–295. DOI: 10.1016/0166-218X(85)90050-2 (cit. on p. 35).

Robin A. Moser and Dominik Scheder (2011). A Full Derandomization of Schöning's k-SAT Algorithm. In: Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing (STOC 2011), pp. 245–252. DOI: 10.1145/1993636.1993670 (cit. on pp. 33, 34).

References

Tobias Nipkow, ed. (2025). Functional Data Structures and Algorithms. A Proof Assistant Approach. Association for Computing Machinery (ACM). DOI: 10.1145/3731369 (cit. on p. 53).

Tobias Nipkow, ed. [2025] (22nd Jan. 2026). Functional Data Structures and Algorithms. A Proof Assistant Approach. Updated version of (Nipkow 2025). URL: https://fdsa-book.net/ (cit. on p. 3).

Dominik Scheder (2008). Guided Search and a Faster Deterministic Algorithm for 3-SAT. In: Proc. of the 8th Latin American Symposium on Theoretical Informatic (LATIN 2008). Ed. by Eduardo Sany Laber, Claudson Bornstein, Tito Loana Nogueira and Luerbio Faria. Vol. 4957. Lecture Notes in Computer Science. Springer, pp. 60–71. DOI: 10.1007/978-3-540-78773-0_6 (cit. on pp. 33, 34).

Ingo Schiermeyer (1996). Pure Literal Look Ahead: An $O(1.497^n)$ 3-Satisfability Algorithm (Extended Abstract). Workshop on the Satisfability Problem, Siena 1996. URL: http://gauss.ececs.uc.edu/franco_files/SAT96/sat-workshop-abstracts.html (cit. on p. 35).

Steven S. Skiena [1997] (2020). The Algorithm Design Manual. 3rd ed. Springer. DOI: 10.1007/978-3-030-54256-6 (cit. on pp. 12, 26, 27).