Proving Divide & Conquer Complexities
in Isabelle/HOL

Manuel Eberl
11th May 2019

Abstract
The Akra–Bazzi method, a generalisation of the well-known Master Theorem, is a useful tool for analysing the complexity of Divide & Conquer algorithms. This work describes a formalisation of the Akra–Bazzi method in the interactive theorem prover Isabelle/HOL and the derivation of the Master Theorem from it.

1 Related Work
The original paper by Akra and Bazzi [1] that introduced the Akra–Bazzi method uses a so-called order transform to reduce the problem to a two-dimensional problem. This version of the method also requires very strong assumptions on the parameters of the problem: the recursive definition must be of the form \( f(x) = g(x) + \sum_{i=1}^{k} a_i f(\lfloor \frac{x}{b_i} \rfloor) \) where \( b_i \in \mathbb{N} \) and \( b_i \geq 2 \) and \( g \) is non-decreasing. In particular, recursive calls like \( f(\lceil \frac{x}{2} \rceil) \), \( f(\lfloor \frac{2}{3}x \rfloor) \), or \( f(\lfloor \frac{2}{3}x \rfloor + 1) \) are not allowed.

Leighton [6] gives a vastly generalised version of the theorem in which the above restrictions on \( g \) and the recursive call are weakened greatly. Furthermore, his approach is much more direct; he gives a simple inductive proof that we deemed much more amenable to formal verification than the original proof by Akra and Bazzi.

Our formal proof in Isabelle/HOL is modelled very closely after Leighton’s. Two assumptions were weakened slightly and another one had to be strengthened considerably to remedy a mistake in Leighton’s proof, although the consequences of this stronger assumption for analysing Divide & Conquer complexities are minimal. We will explain this stronger assumption and its ramifications in detail later.

2 Syntactical note
We take some liberties in presenting expressions or theorems from Isabelle/HOL here to increase readability. In particular: type coercions between real numbers and natural numbers are always omitted; schematic variables, which are implicitely universally qualified in Isabelle, are printed with an explicit \( \forall \) for the sake of clarity; Isabelle-specific syntax, such as \( \{0..<1\} \) is replaced with the standard notation \( [0; 1) \); lists are sometimes implicitly used as sets or as indexed sequences (e.g. \( as_i \) for the \( i \)-th element of \( as \), starting from 1). Note also that in the context of this work, the prime symbol never stands for a derivative. The two functions \( g \) and \( g' \) are simply two functions that have no a-priori connection at all.
3 General setting

For this version of the Akra–Bazzi method, we shall consider a recursively-defined function $f : \mathbb{N} \to \mathbb{R}$ with the following properties:

$$f(x) \geq 0 \quad \text{for all } x \in [x_0; x_1)$$

$$f(x) = g(x) + \sum_{i=1}^{k} a_i f(b_i x + h_i(x)) \quad \text{for all } x \geq x_1$$

for $g : \mathbb{N} \to \mathbb{R}$, $k \in \mathbb{N} \setminus \{0\}$, $x_0, x_1 \in \mathbb{N}$, $a_i \in \mathbb{R}$, $b_i \in \mathbb{R}$, $h_i : \mathbb{N} \to \mathbb{R}$, $p \in \mathbb{R}$ with

- $g(x) \geq 0$ for all $x \geq x_1$
- $a_i \geq 0$ for all $i \in [1; k]$ and $a_i > 0$ for at least one $i \in [1; k]$
- $b_i \in (0; 1)$ for all $i \in [1; k]$
- $\exists \varepsilon > 0. h_i \in o\left(\frac{x}{\ln(x)}\right)$ for all $i \in [1; k]$
- $b_i x + h_i(x) \in \mathbb{N}$ and $x_0 \leq b_i x + h_i(x) < x$ for all $i \in [1; k]$ and all $x \geq x_1$ (well-definedness of $f$)
- $p \geq 0$ and $\sum_{i=1}^{k} a_i b_i^p = 1$

We will now explain the meaning of these variables.

**The function $f$.** The parameters $x_0, x_1, k, a_i, b_i, h_i$ characterise the function $f$. To understand the role of the different parameters, it is useful to look at them in the case when $f$ describes the cost of a Divide & Conquer algorithm: the values between $x_0$ and $x_1$ are the costs of the base cases; the cost of the recursive case is defined recursively as the sum of the costs of the recursive calls and the costs of combining the results of the calls. Each triple $(a_i, b_i, h_i)$ corresponds to $a_i$ recursive calls of the form $b_i x + h_i(x)$; the costs of combining the results are represented by the function $g$.

**Variation terms.** The $h_i$ represent asymptotically small variation terms in the recursive call, allowing some deviation from the linear term $b_i x$.

For example, the terms $f(\lfloor \frac{x}{2} \rfloor)$, $f(\lceil \frac{x}{2} \rceil)$, and $f(\lfloor \frac{x}{2} \rfloor + 42)$ would be admissible, as they can be expressed as $f(b x + h(x))$ for some $b \in (0; 1)$ and some $h : \mathbb{R} \to \mathbb{N}$ where $h \in O(1)$. However, much larger deviations, such as $f(\lfloor n - \sqrt{n} \rfloor)$, are also allowed.

Of course, enough base cases must be provided (i.e. $x_0$ and $x_1$ must be chosen large enough and far enough apart) to fulfil the well-definedness conditions; a function ‘definition’ like $f(x) = f(\lceil \frac{3}{4} x \rceil) + 1$ for $x_1 = 3$ cannot be allowed since $f(3) = f(\lceil \frac{9}{4} \rceil) + 1 = f(3) + 1$ is contradictory.

**The parameter $p$.** The parameter $p$ is uniquely defined by the $a_i$ and $b_i$ and characterises the asymptotic cost of the recursion (independently from the combination costs $g$). To show that $p$ exists and is uniquely defined under the above conditions, we consider the function $t : x \mapsto \sum_{i=1}^{k} a_i b_i^x$.

---

1This is necessary because of the discreteness of the natural numbers, a purely linear term in the recursive call is impossible.
This function is continuous and \( t \to \infty \) and \( t \to 0 \). Therefore, by the intermediate value theorem, a \( p \) such that \( \sum_{i=1}^{k} a_i b_i^p = 1 \) always exists, and since \( t \) is also strictly decreasing, this \( p \) is unique.

The above theorem also demands that \( p \) is non-negative, which is equivalent to \( t(0) = \sum_{i=1}^{k} a_i \geq 1 \) due to the monotonicity of \( t \). The recurrences that occur in Divide & Conquer algorithms typically fulfil that restriction, since the \( a_i \) represent the number of recursive calls. This restriction is present in the original paper by Akra and Bazzi [1], but not in the proof by Leighton [6] that we used as a template. In fact, Leighton specifically applies the theorem to examples with \( p < 0 \), but there is one proof step that implicitly assumes \( p \geq 0 \) and the entire proof does not work for \( p < 0 \).
4 The Akra–Bazzi method

Theorem 1. Fix $g' : \mathbb{R} \to \mathbb{R}$ with $g \in \Omega(g')$, i.e. $g'$ is an asymptotic lower bound for $g$. Assume that:

- $f(x) > 0$ for all sufficiently large $x$
- $g'(x) \geq 0$ for all sufficiently large $x$
- there exist real constants $c > 0$ and $C$ with $\forall i \in [1; k]. C < b_i$ such that for all sufficiently large $x$: $\forall u \in [Cx; x]. g'(u) \leq cg'(x)$
- $g'(x)$ is upper-bounded on any real interval $[a; b]$ for sufficiently large $a$
- $g'(x)/x^{b+1}$ is integrable on any real interval $[a; b]$ for sufficiently large $a$

Then
$$f \in \Omega \left( x^p \left( 1 + \int_t^x \frac{g'(u)}{x^{p+1}} du \right) \right)$$
for any sufficiently large $t$.

Theorem 2. Fix $g' : \mathbb{R} \to \mathbb{R}$ with $g \in O(g')$, i.e. $g'$ is an asymptotic upper bound for $g$. Assume that:

- $g'(x) \geq 0$ for all sufficiently large $x$
- there exist real constants $c > 0$ and $C$ with $\forall i \in [1; k]. C < b_i$ such that for all sufficiently large $x$: $\forall u \in [Cx; x]. g'(u) \geq cg'(x)$
- $g'(x)/x^{b+1}$ is integrable on any real interval $[a; b]$ for sufficiently large $a$

Then
$$f \in O \left( x^p \left( 1 + \int_t^x \frac{g'(u)}{x^{p+1}} du \right) \right)$$
for any sufficiently large $t$.

Combining these two results yields:

Theorem 3. Fix $g' : \mathbb{R} \to \mathbb{R}$ with $g \in \Theta(g')$, i.e. $g'$ has the same asymptotic growth as $g$. Assume that:

- $f(x) > 0$ for all sufficiently large $x$
- $g'(x) \geq 0$ for all sufficiently large $x$
- there exist real constants $c_1, c_2 > 0$ and $C$ with $\forall i \in [1; k]. C < b_i$ such that for all sufficiently large $x$: $\forall u \in [Cx; x]. c_1 g'(x) \leq g'(u) \leq c_2 g'(x)$
- $g'(x)$ is upper-bounded on any real interval $[a; b]$ for sufficiently large $a$
- $g'(x)/x^{b+1}$ is integrable on any real interval $[a; b]$ for sufficiently large $a$

Then
$$f \in \Theta \left( x^p \left( 1 + \int_t^x \frac{g'(u)}{x^{p+1}} du \right) \right)$$
for any sufficiently large $t$. 
5 The Master Theorem

The Akra–Bazzi method is very powerful, but by specialising it for specific functions $g'$, simpler versions of it can be obtained. In the following few paragraphs, we will give some intuitive reasoning for why the Master Theorem with its three cases follows from the Akra–Bazzi method.

Expanding the product in the Akra–Bazzi theorem yields

$$f \in \Theta \left( x^p \right) + \Theta \left( x^p \int_{x_0}^{x} \frac{g'(u)}{x^{p+1}} du \right).$$

The $x^p$ on the left is independent from $g'$ and would still be present even for $g' = 0$; it can therefore be seen as the inherent cost of the recursion itself. The term with the integral on the right becomes large when the recombination costs, i.e. $g'$, become large.

Suppose $g'(x) = x^q$ for some $q \in \mathbb{R}$. Then we have

$$f \in \Theta \left( x^p \right) + \Theta \left( x^p \int_{x_0}^{x} x^{q-p-1} du \right).$$

It is immediately clear that for $q < p$, the right summand will be $\Theta(x^q)$ and thus dominated by the $\Theta(x^p)$ term, yielding an overall $\Theta(x^p)$ (a ‘bottom-heavy recursion’). Conversely, if $q > p$, the right summand dominates the left one, yielding an overall $\Theta(x^q)$ (a ‘top-heavy recursion’). The case of $q = p$ is interesting, since the right summand now becomes

$$\Theta \left( x^p \int_{x_0}^{x} x^{-1} du \right) = \Theta \left( x^p \ln x \right)$$

and again dominates the left summand, but only by a logarithmic factor (a ‘balanced recursion’). This case can be generalised further by letting $g(x) := x^p \ln^q x$.

These informal observations correspond to the three cases of the Master Theorem: $g \in O(x^{p-\varepsilon})$, $g \in \Omega(x^{p+\varepsilon})$, and $g \in \Theta(x^{p+\varepsilon})$.

**Corollary 4** (Master Theorem).

**Bottom-heavy recursion.** If $g \in O(x^q)$ for some $q < p$, then $f \in \Theta(x^p)$. If, additionally, $f(x)$ is positive for all sufficiently large $x$, we even have $f \in \Theta(x^p)$.

**Balanced recursion.** If $g \in \Theta(x^p \ln^q x)$ for some $q$, then

$$f \in \begin{cases} 
\Theta(x^p) & \text{if } q < -1 \\
\Theta(x^p \ln x) & \text{if } q = -1 \\
\Theta(x^p \ln^{q+1} x) & \text{if } q > -1 
\end{cases}$$

**Top-heavy recursion.** If $g \in \Omega(x^q)$ for some $q > p$, then $f \in \Theta(x^q) = \Theta(g)$.

Note that due to the other constraints on $f$ and $g$, the condition that $f(x)$ is positive for all sufficiently large $x$ must hold if either $g(x)$ is positive for all sufficiently large $x$ or $f(x)$ is positive in all the base cases, i.e. $x \in [x_0; x_1)$.
5.1 Comparison with other versions

Due to its being derived from the Akra–Bazzi theorem, this version of the Master’s theorem is much more general than the versions of the Master theorem that are typically presented in the literature (e.g., in Cormen’s Introduction to Algorithms [3]). This version of the Master theorem imposes far fewer restrictions on the shape of the recursive call, allowing multiple terms with floors, ceilings, and other deviations.

Apart from the more general recursion scheme that this version of the Master theorem allows, there are two more differences with the version of the Master’s Theorem given by Cormen [3]:

- The second case of this version of the Master Theorem is more general, as it allows arbitrary real numbers \( q \) whereas the version by Cormen demands \( q \geq 0 \).
- The third case is more restrictive than that by Cormen: we demand \( g \in \Theta(x^q) \), whereas Cormen only demands \( g \in \Omega(x^q) \) and the existence of some \( c < 1 \) such that for all sufficiently large \( x \), the regularity condition \( ag(x/b) \leq cg(x) \) holds. For a more complex recursion scheme, as allowed by our Master theorem, this regularity condition would be much more complicated:

\[
\sum_{i=1}^{k} a_i g(b_i x + h_i(x)) \leq cg(x)
\]

The proof that \( f \) is then in \( \Theta(g) \) is relatively simple and does not require the Akra–Bazzi theorem at all.

---

3 The Master’s theorem presented in the book actually demands \( q = 0 \), but exercise 4.2-2 is a generalisation to \( q \geq 0 \).
6 Formalisation in Isabelle/HOL

6.1 Conditions on the function

In Isabelle/HOL, the conditions mentioned in section 3 are captured in a locale [2]. A locale is a kind of named context that contains fixed variables, assumptions, and definitions. Such a locale can be instantiated by providing values for the fixed variables and proving that its assumptions hold for these values.

The locale akra_bazzi_function from the file Akra_Bazzi.thy corresponds to the conditions stated in section 3. Its definition, modulo some insignificant notational adjustments, is given in Fig. 1.

```isabelle
locale akra_bazzi_function =  
  fixes x :: nat and as bs :: real list and ts :: (nat ⇒ nat) list and f :: nat ⇒ real and g :: nat ⇒ real  
  assumes k ≠ 0 and length as = k and length bs = k and length ts = k  
  and ∀a ∈ as. a ≥ 0 and ∀b ∈ bs. b ∈ (0; 1)  
  and ∀i ∈ [1; k]. akra_bazzi_term x0 x1 bs i ts i  
  and ∀x ∈ [x0; x1]. f x ≥ 0  
  and ∀x ≥ x1. f x = gx + ∑k i=1 as i · f (ts i x)  
  and ∀x ≥ x1. gx ≥ 0  
  and (∑k i=1 as i) ≥ 1
```

Figure 1: The locale akra_bazzi_locale that formally captures the conditions imposed upon the recursively-defined function \( f \)

The only difference to the conditions stated informally before is that the recursive calls are of the shape \( f (ts i x) \) instead of \( f (b i · x + h i x) \). The reason for this is that the latter would require expressing a call like \( f (\lfloor 1/2 x \rfloor) \) in the rather awkward form \( f (\frac{1}{2} x - (\frac{1}{2} x - \lfloor \frac{1}{2} x \rfloor)) \), whereas the former is more direct.

The conditions on the recursive calls are replaced by the condition that all the \( t_i \) be Akra–Bazzi terms, where

```
definition akra_bazzi_term x0 x1 b t =  
  (∃h. ε > 0 ∧ (λx. |h x|) ∈ O(λx. x/ln^{1+ε} x) ∧  
     (∀x ≥ x1. t x ≥ x0 ∧ t x < x ∧ b · x + h x = tx)) .
```

One can then easily prove introduction rules to discharge this condition for specific forms of recursive calls, e.g.

```
lemma akra_bazzi_term_ceiling:  
  assumes b > 0 and b < 1 and x0 ≤ b · x1 and (1 - b) · x1 ≥ 1  
  shows akra_bazzi_term x0 x1 b (λx. ⌈b · x⌉)
```

Provided that such rules exist for every recursive call occurring in the recursive equation of \( f \), this condition, along with most of the other locale assumptions, contain only the constants \( as, bs, k, x_0, \) and \( x_1 \) and can therefore be solved by simple evaluation. The remaining conditions are:
1. \( \forall x \in [x_0; x_1). f(x) \geq 0 \)

2. \( \forall x \geq x_1. g(x) \geq 0 \)

3. \( \forall x \geq x_1. f(x) = g(x) + \sum_{i=1}^{k} a_s i \cdot f(ts_i x) \)

These conditions must be shown by the user, but they are usually direct consequences from the definitions of \( f \) and \( g \).

### 6.2 The Master theorem

From the proof of the Akra–Bazzi theorem in the \texttt{akra_bazzi_function} locale, we can now show the Master theorem, also inside the locale. The user can then interpret the locale for her function \( f \) and use the case of the Master theorem appropriate for her function.

In the Isabelle formalisation, the Master theorem is split into five cases (cf. table 1), with the first case having a weak form \( (O) \) and a strong form \( (\Theta) \).

In Isabelle, it is usually desirable to state the conclusion of a theorem as generally (w.r.t. matching) so that it can be applied in resolution easily. Therefore, the conclusion of case 2.3 is stated not as \( f \in \Theta(x^p \ln^{q+1} x) \), but as \( f \in \Theta(x^p \ln^q x) \), with \( q \) replaced by \( q - 1 \) in the assumptions accordingly. This allows applying the rule to a goal like \( f \in \Theta(x^p \ln^2 x) \) directly, as opposed to the rather awkward \( f \in \Theta(x^p \ln^{1+1} x) \).

<table>
<thead>
<tr>
<th>Case name</th>
<th>Assumptions</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1 ((O))</td>
<td>( g \in O(x^q) ) ( q &lt; p )</td>
<td>( f \in O(x^p) )</td>
</tr>
<tr>
<td>Case 1</td>
<td>( g \in O(x^q) ) ( q &lt; p ) ( f(x) &gt; 0 ) (a.e.)</td>
<td>( f \in \Theta(x^p) )</td>
</tr>
<tr>
<td>Case 2.1</td>
<td>( g \in \Theta(x^p \ln^q x) ) ( q &lt; -1 )</td>
<td>( f \in \Theta(x^p) )</td>
</tr>
<tr>
<td>Case 2.2</td>
<td>( g \in \Theta(x^p / \ln x) )</td>
<td>( f \in \Theta(x^p \ln \ln x) )</td>
</tr>
<tr>
<td>Case 2.3</td>
<td>( g \in \Theta(x^p \ln^{q-1} x) ) ( q &gt; 0 )</td>
<td>( f \in \Theta(x^p \ln^q x) )</td>
</tr>
<tr>
<td>Case 3</td>
<td>( g \in \Theta(x^q) ) ( q &gt; p )</td>
<td>( f \in \Theta(x^q) )</td>
</tr>
</tbody>
</table>

Table 1: The five cases of the Master theorem as formalised in Isabelle/HOL

### 6.3 Automation

The formalisation also contains three proof methods that add a certain degree of automation to the usage of the Master theorem. We will describe them in the following.

#### 6.3.1 Akra–Bazzi terms

As mentioned previously in section 6.1, the condition that recursive calls must be Akra–Bazzi terms can be discharged by introduction rules that reduce the condition to simple statements on constants. The attribute \texttt{akra_bazzi_term_intro} allows the user to add custom introduction rules like this. By adding The attribute \texttt{[akra_bazzi_term_intro]}, the rule is made available to the Akra–Bazzi proof methods, which can then use it to discharge these conditions.
6.3.2 akra_bazzi_termination

Non-primitive recursive definitions in Isabelle/HOL can be done with the function package. For every definition, the user must show that the function is indeed well-defined: the definition must be complete, different equations must not overlap with one another, and there must not be any infinite chains of recursive calls. The last condition, also known as termination, is usually the most difficult to prove. While it can, in many cases, be proven automatically, termination of functions with Akra–Bazzi style recursion can usually not be proven automatically by the function package.

The proof method akra_bazzi_termination uses the akra_bazzi_term_intro rules mentioned before to reduce the proof obligation of termination to simpler proof obligations that contain only constants, which can then be solved automatically using Isabelle’s simplifier. A typical function definition of an Akra–Bazzi function then looks like this:

```isabelle
function merge_sort_cost :: nat ⇒ real where
merge_sort_cost 0 = 0
| merge_sort_cost 1 = 1
| n ≥ 2 ⇒ merge_sort_cost n =
    merge_sort_cost ⌊n/2⌋ + merge_sort_cost ⌈n/2⌉ + n
by force simp_all
termination by akra_bazzi_termination simp_all
```

The invocation ‘by force simp_all’ uses the built-in proof method force and the simplifier to prove that the definition is complete and that the equations do not overlap; the next line proves termination using akra_bazzi_termination, which produces ten proof obligations of the form $0 < \frac{1}{2}, \frac{1}{2} < 1, 0 \leq \frac{1}{2} \cdot 2$, and so on. These are then solved with the simplifier.

6.3.3 master_theorem

The main proof method is master_theorem, which can be invoked on goals of the form $f \in O(\_)$ or $f \in \Theta(\_)$.

- The function $f$ can obviously be determined from the goal itself
- If not provided explicitly, the method will try to obtain the recursive equation for $f$ from the function package, if $f$ was defined with the function package. A heuristic is applied to separate the recursive equation from the base cases.
- $x_0$ is set to 0 if not given explicitly, which is always a correct choice except in the unusual case that $f$ is negative somewhere.
- $x_1$ is determined from the precondition of the recursive equation (e.g. 2 for the precondition $n > 1$).
- $k$, $as$, $bs$, $ts$, and $g$ are always inferred automatically from the recursive equation, provided the required akra_bazzi_term_intro rules exist.
- To provide some more detail as to where all these values come from:

  - The function $f$ can obviously be determined from the goal itself
  - If not provided explicitly, the method will try to obtain the recursive equation for $f$ from the function package, if $f$ was defined with the function package. A heuristic is applied to separate the recursive equation from the base cases.
  - $x_0$ is set to 0 if not given explicitly, which is always a correct choice except in the unusual case that $f$ is negative somewhere.
  - $x_1$ is determined from the precondition of the recursive equation (e.g. 2 for the precondition $n > 1$).
  - $k$, $as$ and $ts$ are determined by partitioning the right-hand side of the recursive equation into summands and each summand into the form $a_i \cdot f(ts_i, n)$ if possible.
  - $g$ is the sum of all summands that cannot be brought into this form.
bs is determined by finding a rule from akra_bazzi_term_intro that matches t for each t in ts.

The proof method also applies some pre-processing and post-processing:

- Goals like $\Theta(\lambda x. x)$ are rewritten to $\Theta(\lambda x. x^1)$ automatically.
- The akra_bazzi_term_intro rules are applied.
- Expressions like $\text{length } [1, 1]$ are evaluated.
- Expressions like $\sum_{i=1}^{2}[1, 1]i \cdot \left[\frac{1}{2}, \frac{1}{4}\right]^2i$ are simplified to $(\frac{1}{2})^2 + (\frac{1}{4})^2$.
- Expressions containing no free variables are solved by simplification unless disabled with the (nosimp) parameter.

This leaves a small number of non-trivial proof obligations for the user to prove.

As an example, we consider the merge_sort_cost function from Section 6.3.2. Case 2.3 of the Master theorem tells us that the growth of this function is $\Theta(n \ln n)$. In Isabelle, we write:

```isar
lemma merge_sort_cost ∈ $\Theta(\lambda n. n \cdot \ln n)$
apply (master_theorem 2.3)
```

This leaves us with the following proof obligations:

\[ \forall x. 0 \leq x \implies x < 2 \implies 0 \leq \text{merge_sort_cost } x \]
\[ \forall x. 2 \leq x \implies \text{merge_sort_cost } x = x + (\text{merge_sort_cost } \lfloor x/2 \rfloor + \text{merge_sort_cost } \lceil x/2 \rceil) \]
\[ \forall x. 2 \leq x \implies 0 \leq x \]
\[ (\lambda x. x) \in \Theta(\lambda x. x \cdot \ln^{1-1} x) \]

After simplification, the only remaining goal is $(\lambda x. x) \in \Theta(\lambda x. x)$, which is easily solved with a corresponding simplification rule.

### 6.4 akra_bazzi_approximation

In some of the cases of the Master theorem, the goal contains the parameter $p$. As shown before, this $p$ always exists and is unique. It can, however, not always be expressed in a closed form. One example for such a situation is the function $f(x) = f(\lfloor x/3 \rfloor) + f(\lceil 3x/4 \rceil)$. In this case, $p$ is a transcendental number that can be approximated to 1.152.

Our Master theorem in Isabelle/HOL yields

```isar
lemma $f \in \Theta(\lambda n. n^{\text{akra_bazzi_exponent } [1, 1] [1/3, 3/4]})$
```

where akra_bazzi_exponent is a function that, given lists $as$ and $bs$ that fulfil the usual conditions, returns the unique $p$ such that

\[ \left(\sum_{i=1}^{k} as_i \cdot bs_i^p\right) = 1 \]

Of course, one would now like to obtain at least an approximate value for this exponent. This can be done with the proof method akra_bazzi_approximate:

```isar
lemma akra_bazzi_exponent [1, 1] [1/3, 3/4] ∈ [1.1519623; 1.1519624] by (akra_bazzi_approximate 29)
```
This proof method internally uses the approximation tactic [4], which uses Taylor series expansions, interval arithmetic, and reflection to certify bounds of the values of certain transcendental functions. The number 29 in the invocation here indicates the precision of the computation. If it is too low, the proof attempt might fail even though the statement is true; if it is too high, the proof might take longer. The user of the proof method must therefore find a precision value that is as low as possible while still being high enough for the proof to succeed.
References


