a formalized checker
for size-optimal sorting networks

luís cruz-filipe
(joint work with peter schneider-kamp)
department of mathematics and computer science
university of southern denmark
types meeting
may 20th, 2015
Sorting networks in a nutshell

Sorting networks, Coq style

Generate-and-prune

Conclusions & future work
A sorting network

This net has 5 channels and 9 comparators.

More info:

The optimal size problem:
What is the minimal number of comparators in a sorting network on $n$ channels ($s_n$)?
**history**

**optimal size**

$s_n$: minimal number of *comparisons* to sort $n$ inputs

<table>
<thead>
<tr>
<th>$n$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_n$</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>12</td>
<td>16</td>
<td>19</td>
<td>25</td>
<td>29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$n$</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_n$</td>
<td>35</td>
<td>39</td>
<td>45</td>
<td>51</td>
<td>56</td>
<td>60</td>
<td>73</td>
</tr>
</tbody>
</table>

values for $n \leq 4$ from information theory

values for $n = 5$ and $n = 7$ by exhaustive case analysis

\[
s_n \geq s_{n-1} + 3 \quad \Rightarrow \quad \text{values for } n = 6, 8
\]

\[
s_n \geq s_{n-1} + \lg(n) \quad \Rightarrow \quad \text{other lower bounds}
\]
**history**

\[ s_n: \text{minimal number of comparisons to sort } n \text{ inputs} \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_n )</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>12</td>
<td>16</td>
<td>19</td>
<td>25</td>
<td>29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( n )</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_n )</td>
<td>35</td>
<td>39</td>
<td>45</td>
<td>51</td>
<td>56</td>
<td>60</td>
<td>73</td>
</tr>
</tbody>
</table>

- generate-and-prune algorithm
- intensive parallel computing
- \( \sim 16 \) years of cpu time to compute \( s_9 \)

but how do we know that these results are correct?
outline

sorting networks in a nutshell

sorting networks, coq style

generate-and-prune

conclusions & future work
pros and cons

the easy stuff

- (very) constructive theory
- everything is decidable
- many proofs by exhaustive case analysis
- elementary definitions

main challenges

- all finite domains (channels, inputs, ...)
- reasoning about permutations (in proofs)
- very informal proofs ("trivial", "exercise", "clearly")
sequence of comparators \((i, j)\) with \(1 \leq i \neq j \leq n\)
n is the number of channels

Definition comparator : Set := (prod nat nat).
Definition comp_net : Set := list comparator.

Definition comp_channels (n:nat) (c:comparator) :=
  let (i,j) := c in (i<n) \(\land\) (j<n) \(\land\) (i<>j).

Definition channels (n:nat) (C:comp_net) :=
  forall c:comparator, (In c C) \rightarrow (comp_channels n c).

\(i < j\) for all \((i, j)\) \(\in\) \(C\)

Definition comp_standard (n:nat) (c:comparator) :=
  let (i,j) := c in (i<n) \(\land\) (j<n) \(\land\) (i<j).

Definition standard (n:nat) (C:comp_net) :=
  forall c:comparator, (In c C) \rightarrow (comp_standard n c).
sorting networks (i/iii)

0/1 lemma
(knuth 1973)

$C$ is a sorting network on $n$ channels iff $C$ sorts all inputs in $\{0, 1\}^n$

Inductive bin_seq : nat -> Set :=
  | empty : bin_seq 0
  | zero : forall n:nat, bin_seq n -> bin_seq (S n)
  | one : forall n:nat, bin_seq n -> bin_seq (S n).

Fixpoint get n (s:bin_seq n) (i:nat) : nat := ...
Fixpoint set n (s:bin_seq n) (i:nat) (x:nat)
  : (bin_seq n) := ...

- similar to Vector from the standard library
- definition of sorted (property) and sort (operation)
- induction principles, exhaustive enumeration
- $\sim$ 70 lemmas in total
sorting networks (ii/iii)

$C(\vec{x})$ denotes the output of $C$ on $\vec{x} = x_1 \ldots x_n$

Fixpoint apply (c:comparator) n (s:bin_seq n) : (bin_seq n) :=
  let (i,j):=c in let x:=(get s i) in let y:=(get s j) in
  match (le_lt_dec x y) with
     | left _ => s
     | right _ => set (set s j x) i y
  end.

Fixpoint full_apply (C:comp_net) n (s:bin_seq n) :
  (bin_seq n) :=
  match C with
     | nil => s
     | cons c C’ => full_apply C’ _ (apply c s)
  end.
sorting networks (ii/iii)

output

$C(\vec{x})$ denotes the output of $C$ on $\vec{x} = x_1 \ldots x_n$

Fixpoint apply (c:comparator) n (s:bin_seq n) : (bin_seq n).

Fixpoint full_apply (C:comp_net) n (s:bin_seq n) : (bin_seq n).

outputs($C$) = \{ $C(\vec{x})$ | $x \in \{0, 1\}^n$ \}

Definition outputs (C:comp_net) (n:nat) : (list (bin_seq n)) := (map (full_apply C (n:=n)) (all_bin_seqs n)).

$C(\vec{x})$ is sorted for every input $\vec{x}$

Definition sort_net (n:nat) (C:comp_net) := (channels n C) /\ 
forall s:bin_seq n, sorted (full_apply C s).

Theorem SN_char : forall C n, channels n C ->
(forall s, In s (outputs C n) -> sorted s) ->
sort_net n C.
**sorting networks (iii/iii)**

---

**sanity check**

Definition SN4 :=  
\[(0[<]1 :: 2[<]3 :: 0[<]2 :: 1[<]3 :: 1[<]2 :: \text{nil})].

Theorem SN4_SN: sort_net 4 SN4.

---

**the bad news**

does not scale for 9 channels

---

**the good news**

“C is a sorting network” is decidable

Lemma SN_dec : forall n C, channels n C ->  
\{\text{sort_net n C}\} + \{\neg\text{sort_net n C}\}.

---

- program extraction \(\leadsto\) haskell program (tests all inputs)
- nearly best possible algorithm (known result)
- short formalization (\(\sim\) 35 lemmas)
the key result

output lemma
(parberry 1991)

if \( \text{outputs}(C) \subseteq \text{outputs}(C') \) and \( C', N \) is a sorting network, then \( C, N \) is a sorting network

permutated output lemma

if \( \pi(\text{outputs}(C)) \subseteq \text{outputs}(C') \) for some permutation \( \pi \) and \( C' \) extends to a sorting network, then \( C \) extends to a sorting network

proof

\[
\begin{align*}
\{0, 1\}^n \xrightarrow{\pi} \{0, 1\}^n & \xrightarrow{C} X \xrightarrow{\text{st}(\pi(N))} S \\
\{0, 1\}^n \xrightarrow{\pi} \{0, 1\}^n & \xrightarrow{C'} X' \xrightarrow{N} S
\end{align*}
\]

\( \sim \sim \) how do we formalize this?
**standardization (i/ii)**

*standardization*

take the first non-standard comparator \((i, j)\) and interchange \(i\) and \(j\) in all subsequent positions; repeat until network is standard

*lemma*

if \(C\) is a sorting network, then so is \(st(C)\)

*proof*

the elements of \(\text{outputs}(st(C))\) are obtained by permuting all elements of \(\text{outputs}(C)\) in the same way; since \(st(C)\) does not change sorted inputs, this permutation must be the identity

\[\rightsquigarrow\text{ in our case: need a (simple?) generalization}\]
Function standardize (C:comp_net) {measure length C} :
  comp_net := match C with
  | nil => nil
  | cons c C' => let (x,y) := c in
    match (le_lt_dec x y) with
    | left _ => (x[<]y :: standardize C')
    | right _ => (y[<]x :: standardize (permute x y C'))
    end
  end.

| not structurally decreasing |
| lots of implicit properties |

Theorem standardization_sort : forall C n,
  sort_net n C -> sort_net n (standardize C).

\[ \Rightarrow \text{requires } \sim 60 \text{ lemmas about permutations} \]
**Definition**

\[ C \preceq_{\pi} C' \text{ if } \pi(\text{outputs}(C)) \subseteq \text{outputs}(C') \]

\[ C \preceq C' \text{ if } C \preceq_{\pi} C' \text{ for some permutation } \pi \]

Variable \( n : \text{nat} \).
Variables \( C, C' : \text{comp_net} \).
Variable \( P : \text{permut} \).
Variable \( HP : \text{permutation } n \ P \).

Definition subsumption :=
\[
\text{forall } s : \text{bin_seq } n, \text{ In } s (\text{outputs } C n) \rightarrow \\
\text{ In } (\text{apply_perm } P s) (\text{outputs } C' n).
\]

Theorem BZ : standard \( n \ C \rightarrow \text{subsumption} \rightarrow \\
\text{sort_net } n (C'++N) \rightarrow \\
\text{sort_net } n (\text{standardize } (C ++ \text{apply_perm_to_net } P N)).
\]

Lemma subsumption_dec : \{\text{subsumption}\} + \{\neg\text{subsumption}\}.
outline

sorting networks in a nutshell

sorting networks, coq style

generate-and-prune

conclusions & future work
the algorithm

**init**

set $R_0^n = \{\emptyset\}$ and $k = 0$

**repeat**

until $k > 1$ and $|R^n_k| = 1$

*generate $N^n_{k+1}$* extend each net in $R^n_k$ by one comparator in all possible ways

*prune to $R^n_{k+1}$* keep only one element from each minimal equivalence class w.r.t. $\leq^T$

*step* increase $k$

**pruning**

- quadratic step
- inner loop searches among all permutations typically fails
- record successful subsumptions
the algorithm

\[ R_0^n = \{\emptyset\} \text{ and } k = 0 \]

until \( k > 1 \) and \( |R_k^n| = 1 \)

\textit{generate} \( N_{k+1}^n \) extend each net in \( R_k^n \) by one comparator in all possible ways

\textit{prune to} \( R_{k+1}^n \) keep only one element from each minimal equivalence class w.r.t. \( \leq^T \)

\textit{step} increase \( k \)

using recorded subsumptions as an oracle

replace pruning cycle by oracle calls

skeptic approach towards oracle

use program extraction

verifies all cases up to \( s_8 \), requires \( \sim 18 \) years for \( s_9 \ldots \)
Definition Oracle := list (comp_net * comp_net * (list nat)).

Inductive Answer : Set :=
| yes : nat -> nat -> Answer
| no : forall n k:nat, forall R:list comp_net,
   NoDup R ->
   (forall C, In C R -> length C = k) ->
   (forall C, In C R -> standard n C) -> Answer
| maybe : Answer.

Fixpoint Generate_and_Prune (m n:nat) (O:list Oracle) :
  Answer.

Theorem GP_no : forall m n O R HR0 HR1 HR2,
  Generate_and_Prune m n O = no m n R HR0 HR1 HR2 ->
  forall C, sort_net m C -> length C > n.

Theorem GP_yes : forall m n O k,
  Generate_and_Prune m n O = yes m k ->
  (forall C, sort_net m C -> length C >= k) \/
  exists C, sort_net m C /\ length C = k.
an offline oracle

typical approach

- call oracle to solve difficult tasks
- check result
- oracle is online, waiting for the next problem

in our case

- oracle is pre-computed (offline)
- information from oracle guides algorithm
- potential for optimizations
**improving the pruning step**

**old algorithm**

while oracle has a next subsumption $C \preceq_{\pi} C'$

1. check that $C \preceq_{\pi} C'$
2. check that $C, C'$ are in the current set
3. remove $C'$ from the current set

(laziness performs the last two steps together)

**new algorithm**

while oracle has a next subsumption $C \preceq_{\pi} C'$

1. check that $C \preceq_{\pi} C'$
2. store $C$
3. remove $C'$ from the current set

after: check that all stored networks are in the final set

**requirement**

cannot have subsumption chains, e.g. $C_1 \preceq C_2 \preceq C_3$
improving the pruning step

new algorithm
1. while oracle has a next subsumption \( C \preceq_\pi C' \)
2. check that \( C \preceq_\pi C' \)
3. store \( C \)
4. remove \( C' \) from the current set

after: check that all stored networks are in the final set

pre-processing
replace chains by endpoint subsumptions (e.g. \( C_1 \preceq C_3 \))

optimizations
- provide \( C' \)'s in the order they were generated (replaces quadratic step by linear)
- replace lists by search trees (improves performance)
- extract naturals to native integers (unfortunately necessary, but clearly sound)
- represent comparators as a single number (reduces memory consumption)
philosophical considerations

the good news

checker verifies $s_9$ in around 6 days using “moderate” resources

not-so-new commonplace cpu, 64 gb ram

(almost) no changes to the formalization

relatively quick changes (a few hours each)

mostly require proving that optimized version coincides with original version

more good news

offline oracles

a new methodology?
outline

sorting networks in a nutshell

sorting networks, coq style

generate-and-prune

conclusions & future work
conclusions & future work

results

- formal verification of exact values of $s_n$ for $n \leq 9$
- new methodology (offline oracles)
- able to deal with $\sim 27$ gb of proof witnesses
- clean separation between formalization ("mathematics") and optimization of checker ("computer science")

next episodes

- formal proof of van voorhis’ $s_n \geq s_{n-1} + \lg(n)$ to obtain $s_{10}$
- other problems in sorting networks
- application of this method to other search-intensive proofs