Formalising Bitonic Sort

Ana Bove

Partly join work with Thierry Coquand

Chalmers University of Technology

December 16th, 2004 – TYPES 2004

Formalising Bitonic Sort Twice!

Ana Bove

Partly join work with Thierry Coquand

Chalmers University of Technology

December 16th, 2004 – TYPES 2004

Content

- Why 2 formalisations of Bitonic Sort?
- Bitonic Sort:
  - A functional algorithm
  - How it works
- Brief description of the 2 formalisations of Bitonic Sort in type theory
- Conclusions: Comparison of both formalisations

Why 2 formalisations of Bitonic Sort?

It started as an example within the *CoVer* group at Chalmers.

- First formalisation proved the sorted property for sequences of *Booleans* and used the *0-1 principle* to extend the result to other ordered sets.

- Second formalisation used *Lattice theory* and *NO* use of the 0-1 principle.

  Based on Thierry's ideas.

What is Bitonic Sort?

It was introduced by Batcher in 1968.

It is one of the fastest sorting algorithm where the sequence of comparisons is not data-dependent

(such algorithms are called *sorting networks*).

It consists of $O(n * log(n)^2)$ comparisons in $O(log(n)^2)$ stages.

This makes sorting networks suitable for implementation in hardware or in parallel processor arrays.

Bitonic Sort Algorithm

Bitonic sort works on sequences of 2^n elements.

Bitonic Sort Algorithm

Bitonic sort works on sequences of 2^n elements.

Haskell implementation that works on complete and balanced trees of height n (von Sydow):

```
data Tree a = Lf a | Bin (Tree a) (Tree a)

reverseT (Lf x) = Lf x
reverseT (Bin l r) = Bin (reverseT r) (reverseT l)
```

Bitonic Sort on Trees

```
sortTree :: Tree a -> Tree a
sortTree (Lf x) = Lf x
sortTree (Bin l r) = merge (Bin (sortTree l)
                                (reverseT (sortTree r)))

merge (Lf x) = Lf x
merge (Bin l r) = Bin (merge l1) (merge r1)
  where (l1,r1) = min-max_Swap l r

min-max_Swap (Lf x) (Lf y) = (Lf (x /\ y), Lf (x \/ y))
min-max_Swap (Bin l1 r1) (Bin l2 r2) = (Bin a c, Bin b d)
   where (a,b) = min-max_Swap l1 l2
         (c,d) = min-max_Swap r1 r2
```

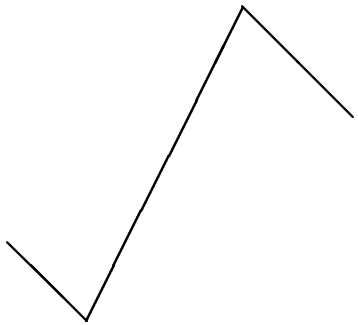
Bitonic Sequence

Definition: A sequence $a_1, a_2, \ldots, a_n$ is *bitonic* if there is a k such that either

$$a_1 \leqslant a_2 \leqslant \cdots \leqslant a_k \geqslant \cdots \geqslant a_n$$

or there is a cyclic shift of the sequence such that this is true.

Bitonic sequences have the general shape of a "*bolt*":



How Does It Work?

How Does It Work?

Main property:

Let $ls * rs$ be a *bitonic* sequence and let $(as, bs) = \texttt{min-max\_Swap}\ ls\ rs$. Then

- as and bs are both bitonic sequences
- $\forall a \in as.\ \forall b \in bs.\ a \leqslant b$

Towards a Formalisation

We work on sequences.

A sequence is a function from a *decidable linear order* to an *ordered set*.

Towards a Formalisation

We work on sequences.

A sequence is a function from a *decidable linear order* to an *ordered set*.

We need to compare the elements in the sequences.

We formalise the ordered sets as *linear orders* and we define the *minimum* ($\wedge$) and *maximum* ($\vee$) operations in the normal way.

Towards a Formalisation

We work on sequences.

A sequence is a function from a *decidable linear order* to an *ordered set*.

We need to compare the elements in the sequences.

We formalise the ordered sets as *linear orders* and we define the *minimum* ($\wedge$) and *maximum* ($\vee$) operations in the normal way.

Main Problem: Not easy to reason about bitonic sequences!

A formalisation of bitonic sort in PVS (Couturier, 1998)

- needed to consider 56 cases!
- lemmas rather incomprehensible

First Formalisation

- We formalise sequences as *dependent binary trees*
- We reduce the linear order (with the elem. to sort) to the set of *Booleans*
- We define *labels* on boolean trees: O, I, OI, IO, OIO, IOI, W
- We use the *0-1 principle* (Knuth, 1973; Bouricius, ca 1954) to extend the result to other ordered sets
- We use the *parametricity theorem* (Reynolds, 1983) to prove the 0-1 principle for our bitonic sort (Day, Launchbury and Lewis, 1999)
- Max. number of cases in the main proofs: 24 (+ 23 empty cases) which can be grouped in 6 main cases.

Second Formalisation

- We formalise sequences as functions from linear orders to linear orders

  (Can be generalised to distributive lattices)

- The sequence $f : I \to D$ is a *bitonic sequence* if

$$\begin{aligned} \forall\, i, j, k, l : I.\quad & i < j \;\&\; j < k \;\&\; k < l \Rightarrow \\ & f(i) \wedge f(k) \leqslant f(j) \vee f(l) \;\& \\ & f(j) \wedge f(l) \leqslant f(i) \vee f(k) \end{aligned}$$

- Max. number of cases in the main proofs: 3 (+ 10 empty cases)!
- Elegant proof!

Conclusions

Comparing the second formalisation to the first one:

- Consider fewer cases then previous formalisations: 3 (+ 10)
  - 24 (+ 23) grouped in 6 subcases using the 0-1 principle in Agda
  - 56 in PVS using general definition of bitonic sequences
- Less intuitive: mainly proving "mathematical" properties
- Very few proofs by induction
- Few "annoying" inequality-reasoning proofs
- Good example for testing Agda/FOL and Agda/Agsy
- Good test case for Agda!