++it Italian C++ Community

www.italiancpp.org

@italiancpp

CPP03 – Functional Techniques in C++

Bartosz Milewski

bartosz@relisoft.com - @BartoszMilewski

http://BartoszMilewski.com/

Grazie a

Sponsor

icubed
Microsoft
intel Software
WPC

aspitalia.com
DomusDotNet
DotDotNet
DotNet Campania
.net hell .it
dotnet Lombardia
DotNetSide
DotNet Toscana
Italian C++ Community
Italian Developer Connection
OrangeDotNet
sharepointcommunity.it
UGIdotNET
UGISS
Visual Basic tips&tricks

What is Programming About?

- Dealing with complexity
  - Decomposing
  - Solving smaller problems
  - Re-composing solutions
- Programming is about *composition*

What is OO Programming About?

- Composition of objects
  - Data hiding
  - Implementation hiding

What’s wrong with OOP?

- Objects don’t compose with *concurrency*
- Recipe for data race:
  - Data hiding + Sharing + Mutation
- Recipe for deadlock:
  - Mutex hiding

Immutability

- Composes with data hiding
- Composes with data sharing
- Introduces no long-distance coupling
- Requires no synchronization
- Functional programming allows *controlled mutation*

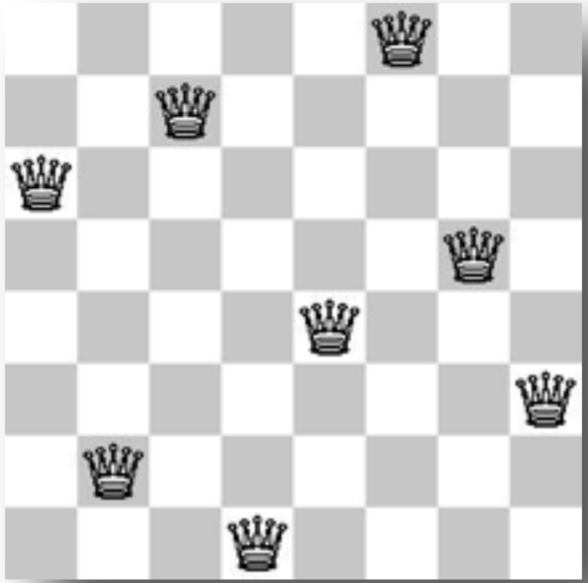
Persistent data structures

- Replace mutation with construction
- Composition of immutable objects
  - Reuse parts in construction
  - Sharing rather than copying
- Old versions *persist*
- *No data race without mutation*

Cautions

- No data is born immutable (publication safety)
- Resource management
  - Shared pointers

- In essence: Safe lock-free data structures



Eight Queens Problem

Persistence in action

Refinement

- Partial solution: k rows with queens
- If k == dim, we are done
- Generate partial solutions with an unchecked queen in k+1st row
- Generic divide and conquer

```
template<class Partial, class Constraint>
std::vector<typename Partial::SolutionT> generate( Partial const & part
                                                 , Constraint constr)
{
    using SolutionVec = std::vector<typename Partial::SolutionT>;

    if (part.isFinished(constr))     {
        SolutionVec result{ part.getSolution() };
        return result;
    }
    else {
        List<Partial> partList = part.refine(constr);

        SolutionVec result;
        forEach(std::move(partList), [&](Partial const & part){
            SolutionVec lst = generate(part, constr);
            std::copy(lst.begin(), lst.end(), std::back_inserter(result));
        });
        return result;
    }
}
```

```
class PartSol // persistent
{
public:
    typedef List<Pos> SolutionT;

    PartSol() : _curRow(0) {}
    PartSol(int row, List<Pos> const & qs)
        : _curRow(row), _queens(qs) {}
    List<Pos> getSolution() const { return _queens; }
    bool isFinished(int dim) const { return _curRow == dim; }
    List<PartSol> refine(int dim) const;
private:
    bool isAllowed(Pos const & pos) const;

    const int _curRow;
    List<Pos> _queens; // persistent
};
```

```
List<PartSol> PartSol::refine(int dim) const
{
    List<PartSol> parts;
    for (int col = 0; col < dim; ++col)
    {
        if (isAllowed(Pos(col, _curRow)))
            parts = parts.push_front(PartSol( _curRow + 1
                                            , _queens.push_front(Pos(col, _curRow))));
    }
    return parts;
}

List List::push_front(T v) const; // Doesn't modify the list!
```

- Persistence vs. backtracking

Parallel Algorithm

```
template<class Partial, class Constraint>
std::vector<typename Partial::SolutionT> generatePar( int depth, Partial const & part
                                                     , Constraint constr)
{
    if (depth == 0)
        return generate(part, constr);
    else if (part.isFinished(constr))
        return { part.getSolution() };
    else {
        List<Partial> partList = part.refine(constr);
        std::vector<std::future<SolutionVec>> futResult;
        forEach(std::move(partList), [&, depth](Partial const & part)
        {
            std::future<SolutionVec> futVec =
                std::async([constr, part, depth]() {
                return generatePar(depth - 1, part, constr);
            });
            futResult.push_back(std::move(futVec));
        });
        std::vector<SolutionVec> all = when_all_vec(futResult);
        return concatAll(all);
    }
}
```

CommunityDays.it

Persistent List

The simplest data structure revisited

A list is...

- Empty, or
- An element (head) and a list (tail)
- This never changes after construction!

A list is...

- Empty, or
- An element (head) and a list (tail)

```
template<class T>
class List // as if we had garbage collection
{
    struct Item {
        Item(T v, Item const * tail) : _val(v), _next(tail) {}
        T _val;
        Item const * _next;
    };
    List() : _head(nullptr) {}
    List(T v, List tail) : _head(new Item(v, tail._head)) {}
    Item const * _head; // null pointer encodes empty list
};
```

```
bool isEmpty() const { return !_head; }

T front() const
{
    assert(!isEmpty());
    return _head->_val;
}

List pop_front() const
{
    assert(!isEmpty());
    return List(_head->_next);
}

List push_front(T v) const
{
    return List(v, *this);
}
```

Memory management

- We don’t have GC!
- We have to use shared_ptr
- Automatically thread safe (but not cheap!)

Memory management

```
template<class T>
class List
{
    struct Item
    {
        Item(T v, std::shared_ptr<const Item> const & tail)
            : _val(v), _next(tail) // <- reference count increased
        {}
        T _val;
        std::shared_ptr<const Item> _next;
    };
public:
    List() {}
    List(T v, List const & tail)
        : _head(std::make_shared<Item>(v, tail._head)) {} // locality!
private:
    std::shared_ptr<const Item> _head;
};
```

CommunityDays.it

Advantages

- Ease of use
  - Implementation follows algorithm
- Composability
- Orthogonality
  - Sequential/Parallel
  - Eager/Lazy

Performance bottlenecks

- Memory allocation/deallocation
- Laziness: Expensive type erasure using std::function
- Random access

Libraries

- https://github.com/BartoszMilewski
- List
- Queue
- Stream
- Red Black Tree (Set and Map)
- Leftist Heap

Q&A

Tutto il materiale di questa sessione su
http://www.communitydays.it/

Lascia il feedback su questa sessione dal sito,
potrai essere estratto per i nostri premi!

Seguici su

Twitter @CommunityDaysIT
Facebook http://facebook.com/cdaysit
#CDays14