Presentation of TC-6

Assistants 2009

May 6, 2014

Presentation of TC-6

Overview of the tarball

The tree structure of TC-6

- New directory:
  - ‘`src/canon`’: Definition of classes for canonization.

The tree structure of TC-6

- New directory:
  - ‘`src/canon`’: Definition of classes for canonization.

Code to write

- Canonicalize the tree: lift ESEQ higher.
- Basic blocks creation.
- Traces creation.
- Other things you need...

Code to write

- Canonicalize the tree: lift ESEQ higher.
- Basic blocks creation.
- Traces creation.
- Other things you need...

Code to write

- Canonicalize the tree: lift ESEQ higher.
- Basic blocks creation.
- Traces creation.
- Other things you need...

Code to write

- Canonicalize the tree: lift ESEQ higher.
- Basic blocks creation.
- Traces creation.
- Other things you need...

C++ notions

Functional programming

Using the STL

Using the STL instead of hand written manipulators, etc. is:

- Cleaner, more readable
- More reliable (the STL is well tested)
- Easier to maintain.

Using the STL

Using the STL instead of hand written manipulators, etc. is:

- Cleaner, more readable
- More reliable (the STL is well tested)
- Easier to maintain.

Using the STL

Using the STL instead of hand written manipulators, etc. is:

- Cleaner, more readable
- More reliable (the STL is well tested)
- Easier to maintain.

Functors

Functors are objects that behave like functions by overloading the `operator()`.
To use functors in the STL algorithms, derive from the STL `std::unary_function` or `std::binary_function`.

```
// This is a predicate, suitable for STL algorithms
// such as std::find_if.
struct block_frontier_p
: public std::unary_function<tree::rTree, bool>
{
  bool
  operator() (const tree::rTree& tree) const
  {
    return (tree.is_a<tree::Label> ()
            || tree.is_a<tree::Jump> ()
            || tree.is_a<tree::Cjump> ());
  }
};
```

Functors as predicates

Functors are generally used to implement predicates.

- Return Booleans
- Action on elements can depend on a predicate.

Functors as predicates

Functors are generally used to implement predicates.

- Return Booleans
- Action on elements can depend on a predicate.

STL algorithms

Many useful algorithms are provided by STL. Here are some examples:

```
// Move items in the range [l.begin (), l.end ())  from l
// to dest. Insertions take place at the beginning of dest.
dest.splice (dest.begin(), l, l.begin (), l.end ());

// Merge sorted containers into dest_sorted_list.
dest_sorted_list.merge (sorted_list);
```

STL algorithms

```
// Returns the first iterator i in the range [first, last)
// such that pred(*i) is true. Returns last if no such
// iterator exists.
find_if (asm.begin (),asm.end (), block_frontier_p ());

// for_each applies the function object f to each element
// in the range [first, last)
template<typename Container>
void
deep_clear (Container& c)
{
  std::for_each (c.begin (), c.end (),
                 Delete<typename Container::value_type> ());
  c.clear ();
}
```

Using Boost::Lambda

Writing functors and use them once is too much work.

- Placeholders: $X = Y * Z$ is equivalent to $\_1 = \_2 * \_3$
- Unnamed functions are so convenient

```
// Print elements in a.
for_each (a.begin (), a.end (), (std::cout << _1 << ' '));

// Print only labels in a.
for_each (a.begin (), a.end (),
          if_(is_a<tree::Label> (_1))[ std::cout << _1 ]);
```

- Bind expressions

```
// Find labels in a.
find_if (a.begin (), a.end (),
         (bind (&is_a<tree::Label>, _1)));
```

Using Boost::Lambda

Writing functors and use them once is too much work.

- Placeholders: $X = Y * Z$ is equivalent to $\_1 = \_2 * \_3$
- Unnamed functions are so convenient

```
// Print elements in a.
for_each (a.begin (), a.end (), (std::cout << _1 << ' '));

// Print only labels in a.
for_each (a.begin (), a.end (),
          if_(is_a<tree::Label> (_1))[ std::cout << _1 ]);
```

- Bind expressions

```
// Find labels in a.
find_if (a.begin (), a.end (),
         (bind (&is_a<tree::Label>, _1)));
```

Using Boost::Lambda

Writing functors and use them once is too much work.

- Placeholders: $X = Y * Z$ is equivalent to $\_1 = \_2 * \_3$
- Unnamed functions are so convenient

```
// Print elements in a.
for_each (a.begin (), a.end (), (std::cout << _1 << ' '));

// Print only labels in a.
for_each (a.begin (), a.end (),
          if_(is_a<tree::Label> (_1))[ std::cout << _1 ]);
```

- Bind expressions

```
// Find labels in a.
find_if (a.begin (), a.end (),
           (bind (&is_a<tree::Label>, _1)));
```