CoCoALib - Bug #1601

Compilation ambiguity

16 Jun 2021 16:28 - John Abbott

Status:	Closed	**Start date:**	16 Jun 2021
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Portability	**Estimated time:**	2.22 hours
Target version:	CoCoALib-0.99800	**Spent time:**	2.15 hours

Description

I have looked better at Florian's problem (in issue #1600).
The cause of the trouble is that the compiler prefers to pick the function ::std::apply rather than ::CoCoA::apply.
Both fns are template functions.

The troublesome line(s) are in HomomorphismOps.C (first is around 147, but other calls to apply trigger trouble too).
An "obvious" use of using ::CoCoA::apply; did not work as hoped.

Related issues:

Related to CoCoALib - Bug #1600: Detect updated versions of external libs	**In Progress**	**14 Jun 2021**
Related to CoCoALib - Design #1467: Change syntax apply(phi,M) into phi(M)?	**Closed**	**22 Jun 2020**
Related to CoCoALib - Design #1576: cmp for machine integers	**In Progress**	**08 Feb 2021**
Related to CoCoALib - Feature #1598: RingHom: implement phi(X) as apply(phi, ...	**Closed**	**10 Jun 2021**

History

#1 - 16 Jun 2021 16:28 - John Abbott

- Related to Bug #1600: Detect updated versions of external libs added

#2 - 16 Jun 2021 16:30 - John Abbott

I am not sure what the problem here is. clang on Anna's computer seems not to have trouble.
My linux box has gcc version 7.5; Florian's machine has gcc 11.1.0.

Might there be a compiler problem? Possible, but unlikely.

The using command above did not cause any compilation warnings or errors, but also did not fix the problem (on Florian's box).

#3 - 21 Jun 2021 14:06 - John Abbott

Right now Florian cannot make further tests... his compiler is defunct (after trying some "gymnastics" to revert to gcc-10).

#4 - 02 Jul 2021 15:41 - John Abbott

- Assignee set to John Abbott

Florian reports problems compiling the following mini-prog using gcc-11.

```
#include <vector>
#include <iostream>
#include <algorithm>

#include <tuple>
using namespace std;

  class BASE
  {
  public:
```

```
    BASE(): datum(0) {}
  private:
    int datum;
  };

  class DERIV: public BASE
  {
  public:
    DERIV(): BASE(), datum2(1) {}
  private:
    int datum2;
  };

  class Function
  {
  public:
    Function() {}
    int operator()(const BASE& b) const { return 2; }
  };

  template <typename T>
  std::vector<int> apply(const Function& F, const std::vector<T>& v)
  {
    std::vector<int> ans;
    std::transform(v.begin(), v.end(), std::back_inserter(ans), [&F](const T& x){ return F(x); });
    return ans;
  }

  int main()
  {
    std::vector<DERIV> v(2,DERIV());
    Function F;
    std::vector<int> vv = apply(F,v);
    std::cout << v.size() << std::endl;
    for (int i: vv)  std::cout << i << std::endl;
  }
```

ERROR message is (in german):

```
In Datei, eingebunden von /usr/include/c++/11.1.0/functional:54,
                 von /usr/include/c++/11.1.0/pstl/glue_algorithm_defs.h:13,
                 von /usr/include/c++/11.1.0/algorithm:74,
                 von test.c:3:
/usr/include/c++/11.1.0/tuple: In Instanziierung von »constexpr const size_t std::tuple_size_v<std::vector<DER
IV> >«:
/usr/include/c++/11.1.0/tuple:1816:24:   erfordert durch »constexpr decltype(auto) std::apply(_Fn&&, _Tuple&&)
 [mit _Fn = Function&; _Tuple = std::vector<DERIV>&]«
test.c:43:32:   von hier erfordert
/usr/include/c++/11.1.0/tuple:1334:61: Fehler: unvollständiger Typ »std::tuple_size<std::vector<DERIV> >« in g
eschachtelter Namensangabe verwendet
 1334 |     inline constexpr size_t tuple_size_v = tuple_size<_Tp>::value;
      |                                                            ^~~~~
```

#5 - 19 Jul 2021 15:54 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

I have now modified **apply.H** so that the **apply** function for vectors is just a normal function (rather than a pointless template fn).
Amazingly, on Florian's computer (Arch Linux with gcc-11.1) the compiler still prefers **std::apply** over the non-template **apply** fn.
What is going on?

Other compilers have no problem!

I do note that cppreference states that std::apply is available from c++17 onwards.
Perhaps Florian needs a flag to limit to the c++14 standard... but that is not a good long-term solution.

Grrr!

#6 - 02 Aug 2021 09:53 - John Abbott

- Related to Design #1467: Change syntax apply(phi,M) into phi(M)? added

#7 - 01 Sep 2021 14:13 - John Abbott

- Related to Design #1576: cmp for machine integers added

#8 - 20 Sep 2021 16:49 - John Abbott

Florian succeeded in installing g++-10 on his computer; and he reports that CoCoALib compiled just fine (except a known problem with test-MachineInt)

Mmmm, what to do?

#9 - 21 Sep 2021 11:31 - John Abbott

I think I know understand what happens: g++ v.11 has as default C++ version something newer than C++14.
My script cxx14.sh simple checks that the version is *at least* C++14, but it should instead check that
the version is *exactly* C++14.

#10 - 30 Sep 2021 14:09 - John Abbott

- Related to Feature #1598: RingHom: implement phi(X) as apply(phi, X) also for X vector and matrix added

#11 - 04 Oct 2021 12:08 - John Abbott

- % Done changed from 10 to 90

This particular problem has been resolved (by eliminating the **apply** fn from CoCoALib).
In this case the removal has led to neater code (I think).

What I do not like is the looming problem of future versions of C++ hiding CoCoALib functions with the same name.
This is a worrying/nasty aspect of new changes introduced into C++; perhaps jumping to C++20, and hoping that
restrictions/constraints on the STL template fn will avoid the problem. Need outside help for this!

#12 - 04 Feb 2022 21:44 - John Abbott

- Status changed from In Progress to Closed

- % Done changed from 90 to 100

- Estimated time set to 2.22 h

Let's cross our fingers and hope that the problem has gone away by the time we jump to C++20.
Closing this issue.... probably a new similar one will appear at some point (sigh).