Subject: Re: Should the pick semantics be changed?
Posted by piotr5 on Wed, 26 Mar 2014 09:35:38 GMT
View Forum Message <> Reply to Message

Lance wrote on Wed, 26 March 2014 02:48it has to do with whether you are constructing with a temporary or a named a varaible, eg

```
struct C{

C(const C& c){}
C(C&& c){}

};

C getC()
{
   C c;
   return std::move(c);
}

int test()
{
   C c=getC(); // the move constructor will be called;
   C*p=new C(getC()); // same move constructor;

   C d=c; // copy ctor will be called;


   C e=std::move(c); // with named variable, std::move or
         // equivalent is required to invoke the move ctor
}
```

yes, that's the example I had in mind. here at the end of getC() the return statement turns c into a temporary since it's getting out of scope after that operation -- very much as in 1st and 2nd line of test(). I'm not saying that behaviour would or wouldn't go against c++11, I just doubt the standard is explicit enough to prevent other compilers from implementing what I said: distinguish between nonstatic local scope variables and all the rest, and just treat those variables which will be destroyed anyway as if they were temporaries, pass them by r-value to the returned class automatically. or can you quote anything from the standard which says that return-values must be initialized by copy-constructor in those cases? if there were a hundred implementations of the c++11 standard, I doubt all 100 would work that way. to convince me otherwise you'd have to quote the definition of what temporaries are and how they differ from values which will be destroyed immediately after this particular statement.

the programmer can't do anything about it, imho changing such behaviour wont destroy compatibility to existing programs, maybe it would improve their performance. and it would definitely remove the burden of optimizing the sourcecode from some of the programmers.

programmers wouldn't need to go through their code, searching for all the return statements, and enclosing the returned objects into "move()" wherever it makes sense! the compiler should take over that optimization task! if gcc is a bit frightened about that change, maybe it should be a compiler-option. but it definitely is needed since programmers cannot influence that except by using automatic code-improvement tools, or code-analysis tools which nag them about some forgotten move() in a return statement...
