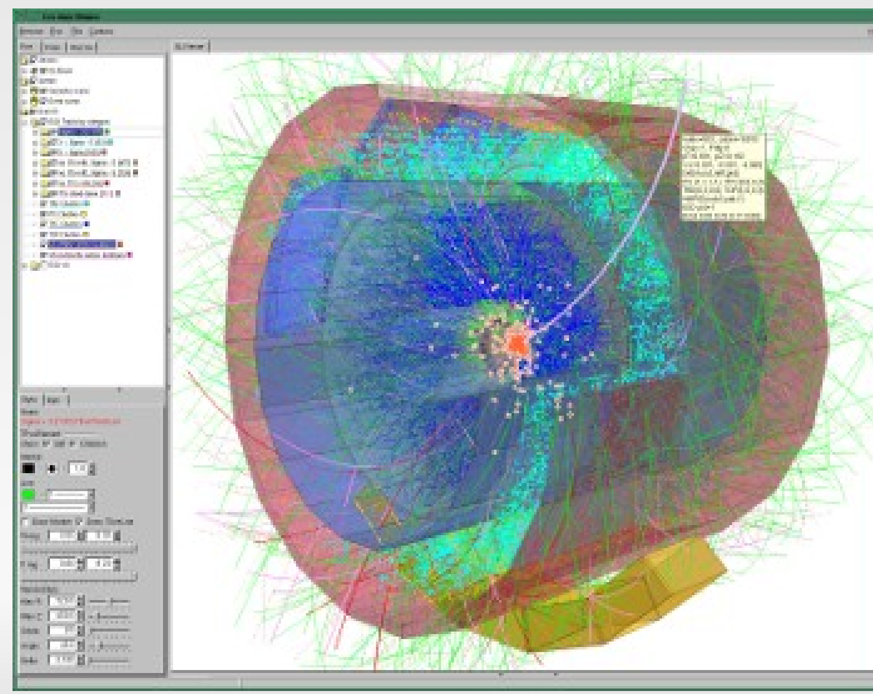
Implementing Dynamic Scopes in Cling

Vassil Vassilev

Domain of High Energy Physics

Use of large scale frameworks and simulators

- Mainly written in C++
- Used by writing C++
- Many non CS users/developers



The ROOT Framework

Efficient data management and analysis

- Toolkit for large scale (PB) data analysis
- About ~20K users
  - Used wherever large data is: HEP, military, banking, astronomy ...
- Huge (~1M LOC)
- Interactive command interface is proven to help not only the newbies but the experts

The ROOT Files

Common storage model used by the experiments

- Serialized C++ objects containing data registered by the experiments
- List of contents (keys): object name, type
- Object data (values)

What is Cling

- C++, C interactive compiler
  - like CsharpRepl (http://www.mono-project.com/CsharpRepl)
  - called ”interpreter” for legacy reasons
- Interactive prompt
  - Terminal-like
  - Allows entering declarations, statements and expressions
- Successor of CINT

Cling Implementation

Cling could be used as library

Built on top of clang and LLVM plus:

- incremental compilation and always incomplete TU
- error recovery
- usability extensions (such as value printing)

Dynamic Scopes in Cling

Extended lookup at runtime

Synopsis

✔ Defined in the root file

```
{
  TFile F;
  if (is_day_of_month_even())
    F.setName("even.root");
  else
    F.setName("odd.root");
  F.Open();
  hist->Draw();
}
hist->Draw();
```

✘ The root file is gone. Issue an error.

Step by Step Plan

During AST construction

```
{
  TFile F;
  if (is_day_of_month_even())
    F.setName("even.root");
  else
    F.setName("odd.root");
  F.Open();
  hist->Draw();
}
hist->Draw();
```

! Failed lookup:
1. Mark the node as dependent. Thus we skip all type checks and continue building the AST

Step by Step Plan

After AST construction

```
{
  TFile F;
  if (is_day_of_month_even())
    F.setName("even.root");
  else
    F.setName("odd.root");
  F.Open();
  hist->Draw();
}
hist->Draw();
```

An ASTConsumer takes care of every dependent node left over and transforms them into valid C++ code

Step by Step Plan

After AST construction

```
{
  TFile F;
  if (is_day_of_month_even())
    F.setName("even.root");
  else
    F.setName("odd.root");
  F.Open();
  EvaluateT<void>("hist->Draw()", ...);
}
hist->Draw();
```

Additional information in case of arguments

Calls cling interface which compiles and runs the dynamic expression

Step by Step Plan

At runtime

```
{
  TFile F;
  if (is_day_of_month_even())
    F.setName("even.root");
  else
    F.setName("odd.root");
  F.Open();
  EvaluateT<void>("hist->Draw()", ...);
}
hist->Draw();
```

```
gCling->Evaluate("hist->Draw()", ...);
```

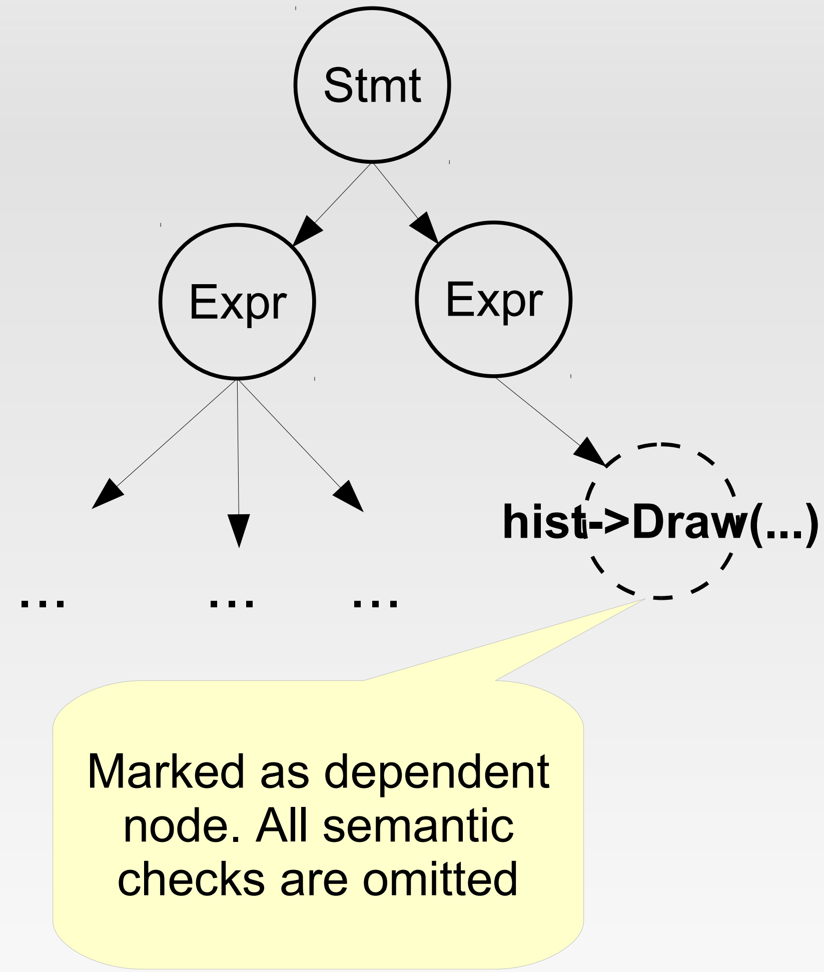
A Real World Example

Functions calls are the most common dynamic expressions in ROOT

```
{
  TFile F;
  F.setName("hist.root");
  F.Open()
  int a[5] = {1, 2, 3, 4, 5};
  int size = 5;
  if (!hist->Draw(a, size))
    return false;
...
}
...
```

AST Transformations

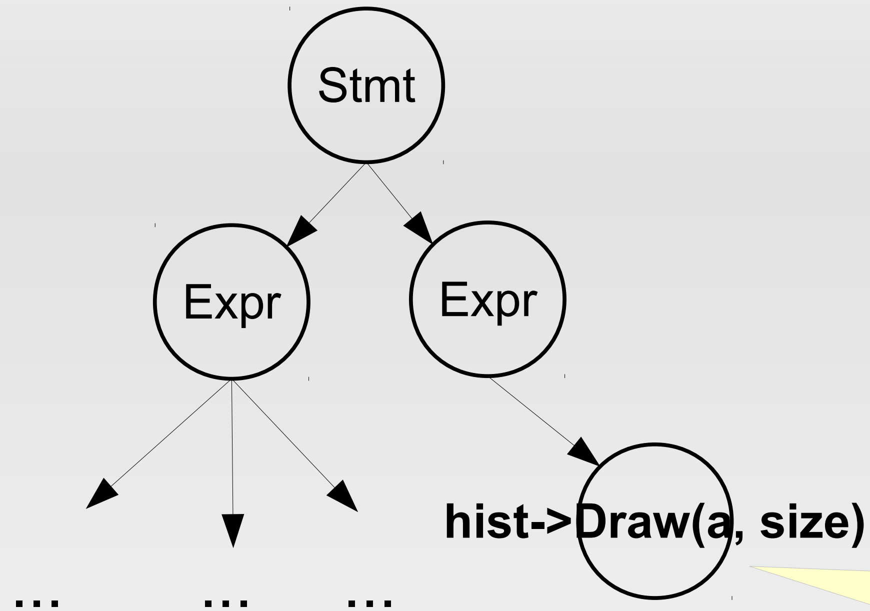
Force Sema to think that it has seen a template definition



Marking every unknown symbol as dependent node is done by overriding the bool LookupUnqualified method in clang's ExternalSemaSource

AST Transformations

Pickup all the artificial nodes "seen as" template definitions



T EvaluateT()

All the information for the subtree's use of other parts of the AST is embedded in the dynamic call site EvaluateT

We need to gather the information about the arguments

hist->Draw(a, size) turns into
bool EvaluateT("**hist->Draw**((int(*))@, *(int*)@)", (void *[2]){ &**a**, &**size** })

Collecting the Relevant Context

In case of more complex expressions (as in previous example) we need to:

- Analyze the subtree that contains the dynamic expression
- Build an extra array of runtime addresses of the used arguments
- “Predict” the expected type of the dynamic expression at compile time

Collecting the Relevant Context

EvaluateT dissected

bool EvaluateT(“hist->Draw((int(*))@, *(int*)@)", (void *[2]){ &a, &size })

Instantiated with the expected return type

Type information

Placeholders, which are replaced by the addresses in the array at runtime

Array of runtime addresses of the relevant context

```
...
  if (!EvaluateT<bool>("hist->Draw((int(*))@, *(int*)@)",
 (void *[2]){ &a, &size }))
...
```

Array of Runtime Addresses

- Needed for the runtime compilation of the dynamic expression
- Artificially generated
- Requires arguments types

```
void*[N]{&arg1, &arg2, ..., &argN}
```

Expected Return Type

```
if (!hist->Draw(a, size))
```

We assume that the entire statement (with return type void) is dynamic unless we've seen an “anchor”, which gives a clue about the expected type.

The dynamic expression was seen in if-clause so we can deduce that the return type of the call site would be bool

Anchor could be:

- Assignment BinOp:
  ```
  int i = hist->Draw(a, size);
  ```
- Explicit cast:
  ```
  (int) hist->Draw(a, size)
  ```
- Implicit cast:
  ```
  if (hist->Draw(a, size))
  ```

Cling's Dynamic Call Site

- EvaluateT
  - Prepare the expression to be fed into cling
  - Returns the expected (T) result
- Evaluate – interface in cling, which:
  - Wraps given dynamic expression
  - Runs the wrapper
  - Returns the result of the execution

Cling's Compiler as Service

Cling provides itself in its environment (gCling)

- Useful for providing an incremental compiler (gCling->processLine(“#include <math>”))
- Used by the dynamic expressions to get compiled at runtime (gCling->Evaluate(“hist->Draw()”))

Unification and Lang Interop

The approach and implementation could be extracted into separate library, as is done for example by DLR (http://dlr.codeplex.com/)

Possible outcome could be:

- Ability cling object to call into library (written in other dynamic language) and dynamically invoke functions on the object that gets back
- Ability dyn lang A to call dyn lang B functions
- Ability to integrate it in other static languages

Demo

1. Load dummy symbol provider (extends the lookup at runtime)
2. Turn on the dynamic expression support
3. Turn on the debug AST printing
4. Type simple dynamic expression

Thank you!