ConceptClang Alpha: Prototype Implementation Notes

Larisse Voufo

Open Systems Lab
Comp. Sci. Program
SOIC, IU-Bloomington, USA

03/17/11
1 Implementation Philosophy

2 The Prototype Implementation Update
ConceptClang: The Goals

1. Implement Concepts in Clang
   - ConceptGCC in a different platform
   - Support all Implementation Design Philosophies:
     - "Indiana" Proposal: "Explicit" Concepts
     - "Texas" Proposal: "Implicit" Concepts
     - "Compromise" Proposal – Pre-Frankfurt Standard
     - "Implicit" Concept w/ "Explicit" Refinements
   - Follow the pre-Frankfurt standard as closely as possible.
   - As first-class entities of the language.
     - Lots of previous work reuse existing features
     - Yet, still no Concept feature.
     - Why not try something different?

2. Analyze issues raised – concretely

3. Determine a right proposal.
ConceptClang: The Goals

1. Implement Concepts in Clang
   - ConceptGCC in a different platform
   - Support all Implementation Design Philosophies:
     - “Indiana” Proposal: “Explicit” Concepts
     - “Compromise” Proposal – Pre-Frankfurt Standard
     - “Implicit” Concept w/ “Explicit” Refinements
   - Follow the pre-Frankfurt standard as closely as possible.
     - As first-class entities of the language.
       - Lots of previous work reuse existing features
       - Yet, still no Concept feature.
       - Why not try something different?

2. Analyze issues raised – concretely

3. Determine a right proposal.
ConceptClang: The Goals

1. Implement Concepts in Clang
   - ConceptGCC in a different platform
   - Support all Implementation Design Philosophies:
     - “Indiana” Proposal: “Explicit” Concepts
     - “Compromise” Proposal – Pre-Frankfurt Standard
     - “Implicit” Concept w/ “Explicit” Refinements
   - Follow the pre-Frankfurt standard as closely as possible.
   - As first-class entities of the language.
     - Lots of previous work reuse existing features
     - Yet, still no Concept feature.
     - Why not try something different?

2. Analyze issues raised – concretely

3. Determine a right proposal.
Motivations

The Fall of Concepts in C++0x

“Not ready, untried, too risky” – paraphrasing Bjarne Stroustrup

- No disagreement on **whether to add** the feature.
- Disagreements on **how to add** the feature.
- Incomplete understanding of implications from each proposal.
- Most of the analysis is abstract and unverified

**Demand for a concrete analysis!**
- Only working prototype: ConceptGCC – insufficient
  - Poor compile-time performance
  - Lack of some advanced features (e.g., scoped concept maps, associated templates)
- Need production-quality implementation
  - to validate the full concepts-based standard library
Motivations

The Fall of Concepts in C++0x

“Not ready, untried, too risky” – paraphrasing Bjarne Stroustrup

- No disagreement on whether to add the feature.
- Disagreements on how to add the feature.
- Incomplete understanding of implications from each proposal.
- Most of the analysis is abstract and unverified
- **Demand for a concrete analysis!**
  - Only working prototype: ConceptGCC – insufficient
    - Poor compile-time performance
    - Lack of some advanced features (e.g., scoped concept maps, associated templates)
  - Need production-quality implementation
    - to validate the full concepts-based standard library
ConceptClang: Features Update

December, 2010
Trivial Concepts, Maps, and Generic Algorithms
- Empty bodies

March, 2011 – Now

1. Features Implemented and Tested
   - Concept definitions (explicit)
   - Concept maps: definitions and instantiation.
   - Associated functions
   - Concept coverage and lookup
   - Concept refinement
   - Associated requirements
   - *late_check
   - Implicit concepts
   - *Explicit refinement
   - Constrained templates: constraints-check
   - Concept ids as qualified name

2. Features Implemented, but Probably Buggy
   - Scoped concepts
   - Associated function templates
   - Concept map templates
   - Associated types

3. In the Horizon:
   1. Most Pressing Features
      - Concept map templates
      - Associated types
      - Concept-based overloading
   2. Eventually
      - Use-patterns
      - Not constraints?
      - etc...
ConceptClang: Features Update

December, 2010
Trivial Concepts, Maps, and Generic Algorithms
- Empty bodies

March, 2011 – Now

1 Features Implemented and Tested
- Concept definitions (explicit)
- Concept maps: definitions and instantiation.
- Associated functions
- Concept coverage and lookup
- Concept refinement
- Associated requirements
- *late_check
- Implicit concepts
- *Explicit refinement
- Constrained templates: constraints-check
- Concept ids as qualified name

2 Features Implemented, but Probably Buggy
- Scoped concepts
- Associated function templates
- Concept map templates
- Associated types

3 In the Horizon:

1 Most Pressing Features
- Concept map templates
- Associated types
- Concept-based overloading

2 Eventually
- Use-patterns
- Not constraints?
- etc...
ConceptClang: Features Update

December, 2010
Trivial Concepts, Maps, and Generic Algorithms
- Empty bodies

March, 2011 – Now

Features Implemented and Tested
1. Concept definitions (explicit)
2. Concept maps: definitions and instantiation.
3. Associated functions
4. Concept coverage and lookup
5. Concept refinement
6. Associated requirements
   *late_check
7. Implicit concepts
8. *Explicit refinement
9. Constrained templates: constraints-check
10. Concept ids as qualified name

Features Implemented, but Probably Buggy
1. Scoped concepts
2. Associated function templates
3. Concept map templates
4. Associated types

In the Horizon:

Most Pressing Features
1. Concept map templates
2. Associated types
3. Concept-based overloading

Eventually
1. Use-patterns
2. Not constraints?
3. etc...
Concepts: The Terminology
... And Main Implementation Checkpoints

**Definition**

```cpp
concept C< typename T > {
    // axiom t = ...
    typename t;
    requires R<T,t>;
    void f(T x, t a);
    ...
}
```

**Model: Concept map**

```cpp
concept_map C<int> {
    typedef int t;
    void f(int x, int a) {... }
    ...
}
```

**Constrained Template**

```cpp
template< typename T >
    requires (C<T>)
    void foo(T x, t a) {
        f(x, a);
    }
```

**Checkpoints**

1. Concept Definition
   - Non-dependent check
2. Concept Map Specification
   - Requirements met?
3. Generic Algorithm Definition
   - Valid concepts?
   - Concept Coverage:
     - Check body against constraint.
4. Generic Algorithm Use
   - Constraints Check:
     - Type matches concept?
     - Pull-in implementation
Concepts: The Terminology

... And Main Implementation Checkpoints

**Definition**

```cpp
class concept C< typename T > {
    // axiom t = ... 
    typedef typename t;
    requires R<T,t>;
    void f(T x, t a);
    ...
}
```

**Model: Concept map**

```cpp
class concept_map C<int> {
    typedef int t;
    void f(int x, int a) {... }
    ...
}
```

**Constrained Template**

```cpp
template< typename T >
    requires (C<T>)
    void foo(T x, t a) {
        f(x, a);
    }
```

**Checkpoints**

1. **Concept Definition**
   - Non-dependent check
2. **Concept Map Specification**
   - Requirements met?
3. **Generic Algorithm Definition**
   - Valid concepts?
   - Concept Coverage:
     - Check body against constraint.
4. **Generic Algorithm Use**
   - Constraints Check:
     - Type matches concept?
     - Pull-in implementation
Concepts: The Terminology
... And Main Implementation Checkpoints

Definition

```cpp
class C< typename T > {
  // axiom t = ...
  typename t;
  requires R<T,t>;
  void f(T x, t a);
  ...
}
```

Model: Concept map

```cpp
concept_map R<int,int> {
  ...
}
```

```cpp
concept_map C<int> {
  typedef int t;
  void f(int x, int a) {... }
  ...
}
```

Checkpoints

1. Concept Definition
   - Non-dependent check

2. Concept Map Specification
   - Requirements met?

3. Generic Algorithm Definition
   - Valid concepts?
   - Concept Coverage:
     - Check body against constraint.

4. Generic Algorithm Use
   - Constraints Check:
     - Type matches concept?
     - Pull-in implementation
Concepts: The Terminology

... And Main Implementation Checkpoints

**Definition**

```cpp
class concept C< typename T > {
    // axiom t = ...
    typename t;
    requires R<T,t>;
    void f(T x, t a);
    ...
};
```

**Model: Concept map Template**

- **Automatic Dispatching**

```cpp
template< typename T >
    requires (R<T,int>)
concept_map C<T> {
    typedef int t;
    void f(T x, int a) { ... }
    ...
}
```

**Checkpoints**

1. **Concept Definition**
   - Non-dependent check
2. **Concept Map Specification**
   - Requirements met?
3. **Generic Algorithm Definition**
   - Valid concepts?
   - Concept Coverage:
     - Check body against constraint.
4. **Generic Algorithm Use.**
   - Constraints Check:
     - Type matches concept?
     - Pull-in implementation
Concepts: The Terminology
... And Main Implementation Checkpoints

Refinement

concept C< typename T > : PC<T> {
   // axiom t = ...
   typename t;
   requires R<T,t>;
   void f(T x, t a);
   ...
}

Model: Concept map

concept_map C<int> {
   typedef int t;
   void f(int x, int a) {... }
   ...
}

Constrained Template

template< typename T >
   requires (C<T>)
   void foo(T x, t a) {
      f(x, a);
   }

Checkpoints

1. Concept Definition
   - Non-dependent check
2. Concept Map Specification
   - Requirements met?
3. Generic Algorithm Definition
   - Valid concepts?
   - Concept Coverage:
     - Check body against constraint.
4. Generic Algorithm Use
   - Constraints Check:
     - Type matches concept?
     - Pull-in implementation
The Prototype Implementation Update

Concepts: The Terminology
... And Main Implementation Checkpoints

Definition

concept C< typename T > : PC<T> {
    // axiom t = ...
    typename t;
    requires R<T,t>;
    void f(T x, t a);
    ...
}

Model: Concept map

concept_map C<int> {
    typedef int t;
    void f(int x, int a) {... }
    ...
}

Constrained Template

template< typename T >
    requires (C<T>)
    void foo(T x, t a) {
    f(x, a);
}

Checkpoints

1. Concept Definition
   - Non-dependent check

2. Concept Map Specification
   - Requirements met?

3. Generic Algorithm Definition
   - Valid concepts?
   - Concept Coverage:
     - Check body against constraint.

4. Generic Algorithm Use.
   - Constraints Check:
     - Type matches concept?
     - Pull-in implementation
Concepts: The Terminology
... And Main Implementation Checkpoints

**Definition**
- associated types
- associated requirements
- associated functions
- Refinement
  - Concept extends requirements of another

**Model: Concept map**
- How a given type meets a concept’s requirements
- (Automatic) Concept Dispatching

**Constrained Template**
- Expressing the constraints on type parameters.

**Checkpoints**
1. Concept Definition
   - Non-dependent check
2. Concept Map Specification
   - Requirements met?
3. Generic Algorithm Definition
   - Valid concepts?
   - Concept Coverage:
     - Check body against constraint.
4. Generic Algorithm Use.
   - Constraints Check:
     - Type matches concept?
     - Pull-in implementation
ConceptClang: Implementation

1. ConceptDecl
   - TemplateDecl, DeclContext
   - TypeParameters
   - Parents
     - Explicit
     - Implicit
   - Requirements
   - Associated Types
     - TemplateTypeParamDecl
     - TemplateTemplateParamDecl
     - TypedefDecl
     - Assigns value to Assoc. Type
   - Associated Functions
     - FunctionDecl
     - FunctionTemplateDecl

2. ConceptMap(Template)Decl
   - TemplateDecl, DeclContext
   - TypeParameters
   - Null ==> ConceptMapDecl
   - TypeArguments
   - ParentMaps
     - Explicit
     - Implicit
   - RequirementMaps
   - Associated Typedefs
     - TypedefDecl
   - Associated Functions
     - FunctionDecl
     - FunctionTemplateDecl

- Concept collect all its maps
  - In a Partial-Ordered Structure.
- 1-to-1 mapping between decls in Concept defns (Declarations) and each one of their maps (Definitions).
ConceptClang: Implementation

1. ConceptDecl
   - TemplateDecl, DeclContext
   - TypeParameters
   - Parents
     - Explicit
     - Implicit
   - Requirements
   - Associated Types
     - TemplateTypeParamDecl
     - TemplateTemplateParamDecl
     - TypedefDecl
   - Associated Functions
     - FunctionDecl
     - FunctionTemplateDecl

2. ConceptMap(Template)Decl
   - TemplateDecl, DeclContext
   - TypeParameters
   - Null ==> ConceptMapDecl
   - TypeArguments
   - ParentMaps
     - Explicit
     - Implicit
   - RequirementMaps
   - Associated Typedefs
     - TypedefDecl
   - Associated Functions
     - FunctionDecl
     - FunctionTemplateDecl

- Concept collect all its maps
- In a Partial-Ordered Structure.
- 1-to-1 mapping between decls in Concept defns (Declarations) and each one of their maps (Definitions).
ConceptClang: Implementation

1. ConceptDecl
   - TemplateDecl, DeclContext
   - TypeParameters
   - Parents
     - Explicit
     - Implicit
   - Requirements
   - Associated Types
     - TemplateTypeParamDecl
     - TemplateTemplateParamDecl
     - TypedefDecl
   - Associated Functions
     - FunctionDecl
     - FunctionTemplateDecl

2. ConceptMap(Template)Decl
   - TemplateDecl, DeclContext
   - TypeParameters
   - TypeArguments
   - ParentMaps
     - Explicit
     - Implicit
   - RequirementMaps
   - Associated Typedefs
     - TypedefDecl
   - Associated Functions
     - FunctionDecl
     - FunctionTemplateDecl

- Concept collect all its maps
- In a Partial-Ordered Structure.
- 1-to-1 mapping between decls in Concept defns (Declarations) and each one of their maps (Definitions).
ConceptClang: Implementation

1. `ConceptDecl`
   - TemplateDecl, DeclContext
   - TypeParameters
   - Parents
     - Explicit
     - Implicit
   - Requirements
   - Associated Types
     - TemplateTypeParamDecl
     - TemplateTemplateParamDecl
     - TypedefDecl
   - Associated Functions
     - FunctionDecl
     - FunctionTemplateDecl

2. `ConceptMap(Template)Decl`
   - TemplateDecl, DeclContext
   - TypeParameters
     - Null \(\Rightarrow\) ConceptMapDecl
   - TypeArguments
   - ParentMaps
     - Explicit
     - Implicit
   - RequirementMaps
   - Associated Typedefs
     - TypedefDecl
   - Associated Functions
     - FunctionDecl
     - FunctionTemplateDecl

- Concept collect all its maps
  - In a Partial-Ordered Structure.
- 1-to-1 mapping between decls in Concept defns (Declarations) and each one of their maps (Definitions).
ConceptClang: Implementation

1. ConceptDecl
   - TemplateDecl, DeclContext
   - TypeParameters
   - Parents
     - Explicit
     - Implicit
   - Requirements
   - Associated Types
     - TemplateTypeParamDecl
     - TemplateTemplateParamDecl
     - TypedefDecl
   - Associated Functions
     - FunctionDecl
     - FunctionTemplateDecl

2. ConceptMap(Template)Decl
   - TemplateDecl, DeclContext
   - TypeParameters
   - Parents
     - Null ==> ConceptMapDecl
   - TypeArguments
   - ParentMaps
     - Explicit
     - Implicit
   - RequirementMaps
   - Associated Typedefs
     - TypedefDecl
   - Associated Functions
     - FunctionDecl
     - FunctionTemplateDecl

- Concept collect all its maps
  - In a Partial-Ordered Structure.
- 1-to-1 mapping between decls in Concept defns (Declarations) and each one of their maps (Definitions).
Conditions

- 1-to-1 mapping between decls in Concept defns (Declarations) and each one of their maps (Definitions).
- ConceptDecl can provide default implementation(s).
- The Rules for looking up definitions:
  - Check in Map.
  - If not, Check in Concept
  - If not, Check in Immediate Surrounding Scope.
- ConceptMapDecl can provide implementations for the associated decls of parents and requirements.
- Reducing verbosity

Parents and Requirements share the same type of Data Structure.

Generating a ConceptMap:
- Maps for its Requirements MUST exist, unless they are for implicit concepts.
- Maps for Parents are implicitly generated, if they don’t exist.
The Prototype Implementation Update

Conditions

- 1-to-1 mapping between decls in Concept defns (Declarations) and each one of their maps (Definitions).
- ConceptDecl can provide default implementation(s).
- The Rules for looking up definitions:
  - Check in Map.
  - If not, Check in Concept
  - If not, Check in Immediate Surrounding Scope.
- ConceptMapDecl can provide implementations for the associated decls of parents and requirements.

Example:

```cpp
int dothis() {... }

concept A<typename T> {
  int dothis();
}

class concept_map<int> {} // Picks up global implementation of dothis()
```
Conditions

- 1-to-1 mapping between decls in Concept defns (Declarations) and each one of their maps (Definitions).
- ConceptDecl can provide default implementation(s).
- The Rules for looking up definitions:
  - Check in Map.
  - If not, Check in Concept
  - If not, Check in Immediate Surrounding Scope.
- ConceptMapDecl can provide implementations for the associated decls of parents and requirements.
  - Reducing verbosity
- Parents and Requirements share the same type of Data Structure.
- Generating a ConceptMap:
  - Maps for its Requirements MUST exist, unless they are for implicit concepts.
  - Maps for Parents are implicitly generated, if they don’t exist.
The Prototype Implementation Update

Conditions

- 1-to-1 mapping between decls in Concept defns (Declarations) and each one of their maps (Definitions).
- ConceptDecl can provide default implementation(s).
- The Rules for looking up definitions:
  - Check in Map.
  - If not, Check in Concept
  - If not, Check in Immediate Surrounding Scope.
- ConceptMapDecl can provide implementations for the associated decls of parents and requirements.
  - Reducing verbosity

- Parents and Requirements share the same type of Data Structure.
- Generating a ConceptMap:
  - Maps for its Requirements MUST exist, unless they are for implicit concepts.
  - Maps for Parents are implicitly generated, if they don’t exist.
Conditions

- 1-to-1 mapping between decls in Concept defns (Declarations) and each one of their maps (Definitions).
- ConceptDecl can provide default implementation(s).
- The Rules for looking up definitions:
  - Check in Map.
  - If not, Check in Concept
  - If not, Check in Immediate Surrounding Scope.
- ConceptMapDecl can provide implementations for the associated decls of parents and requirements.
  - Reducing verbosity
- Parents and Requirements share the same type of Data Structure.
- Generating a ConceptMap:
  - Maps for its Requirements MUST exist, unless they are for implicit concepts.
  - Maps for Parents are implicitly generated, if they don't exist.
Constructing a Concept Map

- If for a valid concept, create a ConceptMapDecl.
- Collect its associated decls into a temporary collection – say \texttt{DeclsInProcess}.
- For each decl in the mapped concept:
  - Lookup the definition in the map.
  - If not found, \texttt{error}.
  - If found, remove from \texttt{DeclsInProcess}.
- For each Requirement in the mapped concept:
  - Type-check
  - Find a map. If not found and concept is implicit, Generate it.
  - Store map in concept map’s RequirementMaps.
- For each Parent in the mapped concept:
  - Type-check
  - Find or Generate a map.
  - Store map in concept map’s ParentMaps.
- If \texttt{DeclsInProcess} is non-empty:
  - If not already processed in refining maps, \texttt{error}.
Constructing a Concept Map

- If for a valid concept, create a ConceptMapDecl.
- Collect its associated decls into a temporary collection – say DeclsInProcess.
- For each decl in the mapped concept:
  - Lookup the definition in the map.
  - If not found, error.
  - If found, remove from DeclsInProcess.
- For each Requirement in the mapped concept.
  - Type-check
  - Find a map. If not found and concept is implicit, Generate it.
  - Store map in concept map’s RequirementMaps.
- For each Parent in the mapped concept.
  - Type-check
  - Find or Generate a map.
  - Store map in concept map’s ParentMaps.
- If DeclsInProcess is non-empty:
  - If not already processed in refining maps, error.
ConceptClang: Features Review

1. Features Implemented and Tested
   - Concept definitions (explicit)
   - Concept maps: definitions and instantiation.
   - Associated functions
   - Concept coverage and lookup
   - Concept refinement
   - Associated requirements
   - *late_check
   - Implicit concepts
   - *Explicit refinement
   - Constrained templates: constraints-check
   - Concept ids as qualified name

2. Features Implemented, but Probably Buggy
   - Scoped concepts
   - Associated function templates
   - Concept map templates
   - Associated types
Constructing a Concept Map – incl. Explicit derivation

- If for a valid concept, create a `ConceptMapDecl`.
- Collect its associated decls into a temporary collection – say `DeclsInProcess`.
- **For each `ExplicitParent` in the mapped concept.**
  - Type-check
  - Find or Generate a map.
  - Store map in concept map’s `ExplicitParentMaps`.
- For each decl in the mapped concept:
  - Lookup the definition in the map.
  - If not found, `error`.
  - If found, remove from `DeclsInProcess`.
- For each Requirement in the mapped concept.
  - Type-check
  - Find a map. If not found and concept is implicit, Generate it.
  - Store map in concept map’s `RequirementMaps`.
- **For each `ImplicitParent` in the mapped concept.**
  - Type-check
  - Find or Generate a map.
  - Store map in concept map’s `ImplicitParentMaps`.
- If `DeclsInProcess` is non-empty:
  - If not already processed in refining maps, `error`. 
ConceptClang: Features Review

1 Features Implemented and Tested

- **Concept definitions** (explicit)
- **Concept maps:** definitions and instantiation.
- **Associated functions**
- Concept coverage and lookup
- **Concept refinement**
- **Associated requirements**
- *late_check
- Implicit concepts
- *Explicit refinement
- Constrained templates: constraints-check
- Concept ids as qualified name

2 Features Implemented, but Probably Buggy

- **Scoped concepts**
- Associated function templates
- Concept map templates
- Associated types
ConceptClang: Implementation

3. TemplateDecl Specification

- Collect required concepts
  - Type-check each against templates parameters
  - Generate **concept map archetypes** for each.
  - Collect archetypes in current scope.

- Concept Coverage
  - Check body of algorithm against required concepts (their map archetypes).
Constructing a Concept Map: Generating a Concept Map Archetype

- If for a valid concept, create a ConceptMapDecl or ConceptMapArchetype.
- Collect its associated decls into a temporary collection – say DeclsInProcess.
- For each ExplicitParent in the mapped concept:
  - Type-check
  - Find or Generate a map.
  - Store map in concept map’s ExplicitParentMaps.
- For each decl in the mapped concept:
  - If isArchetype:
    - Copy decl’s prototype. Substitute types.
  - Otherwise:
    - Lookup the definition in the map.
    - If not found, **error**.
    - If found, remove from DeclsInProcess.
- For each Requirement in the mapped concept:
  - Type-check
  - Find a map. If not found and concept is implicit, Generate it.
  - Store map in concept map’s RequirementMaps.
- For each ImplicitParent in the mapped concept:
  - Type-check
  - Find or Generate a map.
  - Store map in concept map’s ImplicitParentMaps.
- If DeclsInProcess is non-empty:
  - If not already processed in refining maps, error.
Concept Coverage

- New scope kinds: RestrictedScope
  - At occurrence of **requires** keyword.
- Extension to current lookup procedure:
  - If in RestrictedScope:
    - lookup in concept map archetypes.
    - Exceptions: TemplateParamScope, LateCheckScope, Non-dependent CallExpr, ...
  - If in LateCheckScope:
    - proceed as usual, looking into archetypes as well
  - Lookup of Non-dependent CallExpr:
    - add LateCheckScope to scope flags
  - Lookup of other allowed expressions:
    - ... Work In Progress ...
Concept Coverage and late_check

- New scope kinds: RestrictedScope, LateCheckScope
  - At occurrence of requires keyword.
- Extension to current lookup procedure:
  - If in RestrictedScope:
    - lookup in concept map archetypes.
    - Exceptions: TemplateParamScope, LateCheckScope, Non-dependent CallExpr, ...
  - If in LateCheckScope:
    - proceed as usual, looking into archetypes as well
- Lookup of Non-dependent CallExpr:
  - add LateCheckScope to scope flags
- Lookup of other allowed expressions:
  - ... Work In Progress ...
Concept Coverage and late_check

- New scope kinds: RestrictedScope, LateCheckScope
  - At occurrence of requires keyword.
- Extension to current lookup procedure:
  - If in RestrictedScope:
    - lookup in concept map archetypes.
    - Exceptions: TemplateParamScope, LateCheckScope, Non-dependent CallExpr, ...
  - If in LateCheckScope:
    - proceed as usual, looking into archetypes as well
  - Lookup of Non-dependent CallExpr:
    - add LateCheckScope to scope flags
  - Lookup of other allowed expressions:

Example: Non-dependent CallExpr

```cpp
concept A<typename T> {
    int f(T);
}

template<typename T>
requires A<T>
void myfunc(T a, T b) {
    f(a) == f(b);   // call to '==' is non-dependent.
}
```
Concept Coverage and late_check

- New scope kinds: RestrictedScope, LateCheckScope
  - At occurrence of `requires` keyword.
- Extension to current lookup procedure:
  - If in RestrictedScope:
    - lookup in concept map archetypes.
    - Exceptions: TemplateParamScope, LateCheckScope, Non-dependent CallExpr, ...
  - If in LateCheckScope:
    - proceed as usual, looking into archetypes as well
- Lookup of Non-dependent CallExpr:
  - add LateCheckScope to scope flags
- Lookup of other allowed expressions:

Example: Non-dependent CallExpr

```cpp
class A<typename T> {  
    int f(T);
};

template<typename T>  
    requires A<T>  
void myfunc(T a, T b) {  
    f(a) == f(b);  // call to '==' is non-dependent.
}
```
Concept Coverage and late_check

- New scope kinds: RestrictedScope, LateCheckScope
  - At occurrence of **requires** keyword.
- Extension to current lookup procedure:
  - If in RestrictedScope:
    - lookup in concept map archetypes.
    - Exceptions: TemplateParamScope, LateCheckScope, Non-dependent CallExpr, ...
  - If in LateCheckScope:
    - proceed as usual, looking into archetypes as well
  - Lookup of Non-dependent CallExpr:
    - add LateCheckScope to scope flags

**Example: late_check**

```cpp
concept A<typename T> { 
    T f(T);
}
template<typename T>
    requires A<T>
void myfunc(T a, T b) {
    late_check {
        f(a) == f(b); // call to '==' is not non-dependent. Would not work without late_check.
    }
}
```

Larisse Voufo (Open Systems Lab Comp. Sci. Program SOIC, IU-Bloomington, USA)
The Prototype Implementation Update

Concept Coverage and late_check

- New scope kinds: RestrictedScope, LateCheckScope
  - At occurrence of `requires` keyword.
- Extension to current lookup procedure:
  - If in RestrictedScope:
    - lookup in concept map archetypes.
    - Exceptions: TemplateParamScope, LateCheckScope, Non-dependent CallExpr, ...
  - If in LateCheckScope:
    - proceed as usual, looking into archetypes as well
  - Lookup of Non-dependent CallExpr:
    - add LateCheckScope to scope flags
  - Lookup of other allowed expressions:
    - ... Work In Progress ...
ConceptClang: Features Review

1 Features Implemented and Tested
   - Concept definitions (explicit)
   - Concept maps: definitions and instantiation.
   - Associated functions
   - Concept coverage and lookup
   - Concept refinement
   - Associated requirements
   - *late_check
   - Implicit concepts
   - *Explicit refinement
   - Constrained templates: constraints-check
   - Concept ids as qualified name

2 Features Implemented, but Probably Buggy
   - Scoped concepts
   - Associated function templates
   - Concept map templates
   - Associated types
4. **TemplateDecl Use**

- **Type-check**
  - Template arguments against parameters

- **Constraints-check**
  - Template arguments and parameters against each required concept
  - find or generate maps for each required concept.

- **Create TemplateDecl specialization**
  - Mark for instantiation.

- **Instantiate specialization**
  - Comes much later… At end of translation unit.
4. **TemplateDecl Use**

- **Type-check**
  - Template arguments against parameters

- **Constraints-check**
  - Template arguments and parameters against each required concept
  - find or generate maps for each required concept.

- Create TemplateDecl specialization
  - Mark for instantiation.

- Instantiate specialization
  - Comes much later... At end of translation unit.
DeclContext must be instantiated by now.
- Get body from template decl.
- Perform necessary substitutions / transformations.
  - Transform each statement/decl in body.
- If stmt/decl refers to a member of a concept:
  - If is CallExpr:
    - Identify Concept.
    - Find map for template arguments.
    - Mark map for instantiation
    - Rebuild CallExpr, looking up in identifier in map.
  - If is any Decl:
    - Identify Concept.
    - Find map for template arguments.
    - Find Decl’s definition in Map.
    - Mark map for instantiation. Also Mark Decl if necessary.
    - Rebuild CallExpr, looking up in identifier in map.
- Ideally, this procedure can be re-used for references to types.
  - But it currently does not work.
  - At this point, Clang’s structure for types do not give access to DeclContext
  - Work in progress...
Instantiating a Specialization

- DeclContext must be instantiated by now.
- Get body from template decl.
- Perform necessary substitutions / transformations.
  - Transform each statement/decl in body.
- If stmt/decl refers to a member of a concept:
  - If is CallExpr:
    - Identify Concept.
    - Find map for template arguments.
    - Mark map for instantiation
    - Rebuild CallExpr, looking up in identifier in map.
  - If is any Decl:
    - Identify Concept.
    - Find map for template arguments.
    - Find Decl’s definition in Map.
    - Mark map for instantiation. Also Mark Decl if necessary.
    - Rebuild CallExpr, looking up in identifier in map.
- Ideally, this procedure can be re-used for references to types.
  - But it currently does not work.
  - At this point, Clang’s structure for types do not give access to DeclContext
  - Work in progress...
Instantiating a Specialization

- DeclContext must be instantiated by now.
- Get body from template decl.
- Perform necessary substitutions / transformations.
  - Transform each statement/decl in body.
- If stmt/decl refers to a member of a concept:
  - If is CallExpr:
    - Identify Concept.
    - Find map for template arguments.
    - Mark map for instantiation
    - Rebuild CallExpr, looking up in identifier in map.
  - If is any Decl:
    - Identify Concept.
    - Find map for template arguments.
    - Find Decl’s definition in Map.
    - Mark map for instantiation. Also Mark Decl if necessary.
    - Rebuild CallExpr, looking up in identifier in map.
- Ideally, this procedure can be re-used for references to types.
  - But it currently does not work.
  - At this point, Clang’s structure for types do not give access to DeclContext.
  - Work in progress...
Instantiating a Specialization

- DeclContext must be instantiated by now.
- Get body from template decl.
- Perform necessary substitutions / transformations.
  - Transform each statement/decl in body.
- If stmt/decl refers to a member of a concept:
  - If is CallExpr:
    - Identify Concept.
    - Find map for template arguments.
    - Mark map for instantiation
    - Rebuild CallExpr, looking up in identifier in map.
  - If is any Decl:
    - Identify Concept.
    - Find map for template arguments.
    - Find Decl’s definition in Map.
    - Mark map for instantiation. Also Mark Decl if necessary.
    - Rebuild CallExpr, looking up in identifier in map.
- Ideally, this procedure can be re-used for references to types.
  - But it currently does not work.
  - At this point, Clang’s structure for types do not give access to DeclContext.
  - Work in progress...
Instantiating a Specialization

- DeclContext must be instantiated by now.
- Get body from template decl.
- Perform necessary substitutions / transformations.
  - Transform each statement/decl in body.
- If stmt/decl refers to a member of a concept:
  - If is CallExpr:
    - Identify Concept.
    - Find map for template arguments.
    - Mark map for instantiation
    - Rebuild CallExpr, looking up in identifier in map.
  - If is any Decl:
    - Identify Concept.
    - Find map for template arguments.
    - Find Decl’s definition in Map.
    - Mark map for instantiation. Also Mark Decl if necessary.
    - Rebuild CallExpr, looking up in identifier in map.
- Ideally, this procedure can be re-used for references to types.
  - But it currently does not work.
  - At this point, Clang’s structure for types do not give access to DeclContext.
  - Work in progress...
Instantiating a Specialization

- Option 1: Generate maps at each reference point.
- Option 2: Generate maps once.
- Solution: Option 2
4. **TemplateDecl Use – Update**

- **Type-check**
  - Template arguments against parameters
- **Constraints-check**
  - Template arguments and parameters against each required concept
  - find or generate maps for each required concept.
  - **Collect maps in temporary collection.**
- Create TemplateDecl specialization
  - **Store generated maps in specialization.**
  - Mark for instantiation.
- Instantiate specialization
  - Comes much later... At end of translation unit.
Instantiating a Specialization

- DeclContext must be instantiated by now.
- Get body from template decl.
- Perform necessary substitutions / transformations.
  - Transform each statement/decl in body.

- If stmt/decl refers to a member of a concept:
  - If is CallExpr:
    - Identify Concept.
    - Find map for template arguments – in specialization’s maps.
    - Mark map for instantiation
    - Rebuild CallExpr, looking up in identifier in map.
  - If is any Decl:
    - Identify Concept.
    - Find map for template arguments – in specialization’s maps.
    - Find Decl’s definition in Map.
    - Mark map for instantiation. Also Mark Decl if necessary.
    - Rebuild CallExpr, looking up in identifier in map.

- If is Class specialization:
  - Propagate RequirementMaps from specialization to inner decls.

Ideally, this procedure can be re-used for references to types.
- But it currently does not work.
- At this point, Clang’s struture for types do not give access to DeclContext.
- Work in progress...
Constraints-check procedure

Given:

- TemplateParams, TemplateArgs,
- RequiredConcepts, RequiredConceptMaps

For each RequiredConcept:

- Identify:
  - RequiredConceptParams, RequiredConceptArgs

Match RequiredConceptArgs against TemplateParams and TemplateArgs:

- produces RequiredConceptMapArgs

Try finding a map matching RequiredConceptMapArgs — say RequiredConceptMap.

If found, add RequiredConceptMap to RequiredConceptMaps

Else if RequiredConcept is implicit, Generate the map — RequiredConceptMap

If success, add to RequiredConceptMaps

Else, FAIL!

Constrained Template Definition

```cpp
template< typename T ... >
  requires (C<T> ... )
  void foo(T x, ... , int a) {
    f(x, a);
  }
```

Constrained Template Use

```cpp
foo<char ... >('a', 1);
```
Constraints-check procedure

- Given:
  - TemplateParams, TemplateArgs,
  - RequiredConcepts, RequiredConceptMaps
- For each RequiredConcept:
  - Identify:
    - RequiredConceptParams, RequiredConceptArgs
  - Match RequiredConceptArgs against TemplateParams and TemplateArgs:
    - produces RequiredConceptMapArgs
  - Try finding a map matching RequiredConceptMapArgs – say RequiredConceptMap
    - if found, add RequiredConceptMap to RequiredConceptMaps
    - else if RequiredConcept is implicit, Generate the map – RequiredConceptMap
    - if success, add to RequiredConceptMaps
    - else, FAIL!

Constrained Template Definition

```cpp
template< TemplateParam ... >
  requires (RequiredConcept<T> ... )
  void foo(TemplateParam x, ... , int a) {
    f(x, a);
  }
```

Constrained Template Use

```cpp```
foo<TemplateArg ... >(’a’, 1);
```cpp```
Constraints-check procedure

- Given:
  - `TemplateParams`, `TemplateArgs`, `RequiredConcepts`, `RequiredConceptMaps`
- For each `RequiredConcept`:
  - Identify:
    - `RequiredConceptParams`, `RequiredConceptArgs`
  - Match `RequiredConceptArgs` against `TemplateParams` and `TemplateArgs`:
    - produces `RequiredConceptMapArgs`
  - Try finding a map matching `RequiredConceptMapArgs` – say `RequiredConceptMap`.
  - if found, add `RequiredConceptMap` to `RequiredConceptMaps`
  - else if `RequiredConcept` is implicit,
    - Generate the map – `RequiredConceptMap`.
    - if success, add to `RequiredConceptMaps`.
    - else, FAIL!
Constraints-check procedure

- Given:
  TemplateParams, TemplateArgs, RequiredConcepts, RequiredConceptMaps

- For each RequiredConcept:
  - Identify:
    RequiredConceptParams, RequiredConceptArgs.
  - Match RequiredConceptArgs against TemplateParams and TemplateArgs:
    produces RequiredConceptMapArgs.
  - Try finding a map matching RequiredConceptMapArgs – say RequiredConceptMap.
    - if found, add RequiredConceptMap to RequiredConceptMaps
    - else if RequiredConcept is implicit,
      - Generate the map – RequiredConceptMap.
    - if success, add to RequiredConceptMaps.
    - else, FAIL!
Constraints-check procedure

Given:

- TemplateParams, TemplateArgs, RequiredConcepts, RequiredConceptMaps

For each RequiredConcept:

- Identify:
  - RequiredConceptParams, RequiredConceptArgs.
- Match RequiredConceptArgs against TemplateParams and TemplateArgs:
  - produces RequiredConceptMapArgs.
- Try finding a map matching RequiredConceptMapArgs – say RequiredConceptMap.
- if found, add RequiredConceptMap to RequiredConceptMaps
- else if RequiredConcept is implicit,
  - Generate the map – RequiredConceptMap.
  - if success, add to RequiredConceptMaps.
  - else, FAIL!
Constraints-check procedure

- Given:
  - TemplateParams, TemplateArgs,
  - RequiredConcepts, RequiredConceptMaps
- For each RequiredConcept:
  - Identify:
    - RequiredConceptParams, RequiredConceptArgs.
  - Match RequiredConceptArgs against TemplateParams and TemplateArgs:
    - produces RequiredConceptMapArgs.
  - Try finding a map matching RequiredConceptMapArgs – say RequiredConceptMap.
  - if found, add RequiredConceptMap to RequiredConceptMaps
  - else if RequiredConcept is implicit, 
    - Generate the map – RequiredConceptMap.
    - if success, add to RequiredConceptMaps.
    - else, FAIL!

Also applies to Concept Map Generation!
Constraints-check procedure – For Concept Map Generation

Given:

ConceptParams, ConceptMapArgs, Parents/Requirements, ParentMaps/RequirementMaps

For each Parent/Requirement:

Identify:


Match *Args against ConceptParams and ConceptMapArgs:

produces *MapArgs.

Try finding a map matching *MapArgs – say ParentMap/RequirementMap.

if found, add ParentMap/RequirementMap to ParentMaps/RequirementMaps

else if Parent or Requirement is implicit

Generate the map – ParentMap/RequirementMap.

if success, add to ParentMaps/RequirementMaps

else, FAIL!

Concept Definition

class A< typename T ... > : PA<T> ... {
    ...
    requires (C<T> ... )
    ...
}

Concept Map

concept_map A < char ... > {
    ...
}
Constraints-check procedure – For Concept Map Generation

- **Given:**
  - `ConceptParams`, `ConceptMapArgs`, `Parents/Requirements`, `ParentMaps/RequirementMaps`
- For each `Parent/Requirement`:
  - Match `*Args` against `ConceptParams` and `ConceptMapArgs`:
    - Produces `*MapArgs`.
  - Try finding a map matching `*MapArgs` – say `ParentMap/RequirementMap`.
    - If found, add `ParentMap/RequirementMap` to `ParentMaps/RequirementMaps`.
    - Else if `Parent` or `Requirement` is implicit, generate the map – `ParentMap/RequirementMap`.
      - If success, add to `ParentMaps/RequirementMaps`.
      - Else, **FAIL!**

**Concept Definition**

```plaintext
class A< ConceptParam ... > : Parent<T> ... {
    ...
    requires (Requirement<T> ... )
    ...
}
```

**Concept Map**

```plaintext
class concept_map A < ConceptMapArg ... > {
    ...
}
```
Constraints-check procedure – For Concept Map Generation

Given:

- `ConceptParams`, `ConceptMapArgs`, `Parents/Requirements`, `ParentMaps/RequirementMaps`

For each `Parent/Requirement`:

- Identify:
- Match `*Args` against `ConceptParams` and `ConceptMapArgs`:
  - produces `*MapArgs`.
- Try finding a map matching `*MapArgs` – say `ParentMap/RequirementMap`.
- if found, add `ParentMap/RequirementMap` to `ParentMaps/RequirementMaps`
- else if `Parent` or `Requirement` is implicit
  - Generate the map – `ParentMap/RequirementMap`.
  - if success, add to `ParentMaps/RequirementMaps`.
  - else, FAIL!
Constraints-check procedure – For Concept Map Generation

Given:

- ConceptParams, ConceptMapArgs,
- Parents/Requirements, ParentMaps/RequirementMaps

For each Parent/Requirement:

- Identify:
  
  
  Match *Args against ConceptParams and ConceptMapArgs:
    
    - produces *MapArgs.
  
- Try finding a map matching *MapArgs – say ParentMap/RequirementMap.
  
  - if found, add ParentMap/RequirementMap to ParentMaps/RequirementMaps
  
  - else if Parent or Requirement is implicit

    - Generate the map – ParentMap/RequirementMap.
    
    - if success, add to ParentMaps/RequirementMaps.
    
    - else, FAIL!
Constructing a Concept Map

- If for a valid concept, create a ConceptMapDecl or ConceptMapArchetype.
- Collect its associated decls into a temporary collection – say `DeclsInProcess`.
- **For each ExplicitParent in the mapped concept.**
  - Type-check
  - Find or Generate a map.
  - Store map in concept map’s ExplicitParentMaps.
- For each decl in the mapped concept:
  - If isArchetype:
    - Copy decl’s prototype. Substitute types.
  - Otherwise:
    - Lookup the definition in the map.
    - If not found, error.
    - If found, remove from `DeclsInProcess`.
- **For each Requirement in the mapped concept.**
  - Type-check
  - Find a map. If not found and concept is implicit, Generate it.
  - Store map in concept map’s RequirementMaps.
- **For each ImplicitParent in the mapped concept.**
  - Type-check
  - Find or Generate a map.
  - Store map in concept map’s ImplicitParentMaps.
- If `DeclsInProcess` is non-empty:
  - If not already processed in refining maps, error.
Constructing a Concept Map

- If for a valid concept, create a ConceptMapDecl or ConceptMapArchetype.
- Collect its associated decls into a temporary collection – say **DeclsInProcess**.
- **Constraint-check:**
  - Mapped concept’s parameters, map’s arguments,
  - Mapped concept’s ExplicitParents, ExplicitParentMaps

- For each decl in the mapped concept:
  - If isArchetype:
    - Copy decl’s prototype. Substitute types.
  - Otherwise:
    - Lookup the definition in the map.
    - If not found, **error**.
    - If found, remove from **DeclsInProcess**.

- **Constraint-check:**
  - Mapped concept’s parameters, map’s arguments,
  - Mapped concept’s Requirements, RequirementMaps

- **Constraint-check:**
  - Mapped concept’s parameters, map’s arguments,
  - Mapped concept’s ImplicitParents, ImplicitParentMaps

- If **DeclsInProcess** is non-empty:
  - If not already processed in refining maps, **error**.
The Constraints-check procedure

- Given:
  - TemplateParams, TemplateArgs,
  - RequiredConcepts, RequiredConceptMaps
- For each RequiredConcept:
  - Identify:
    - RequiredConceptParams, RequiredConceptArgs.
  - Match RequiredConceptArgs against TemplateParams and TemplateArgs:
    - produces RequiredConceptMapArgs.
  - Try finding a map matching RequiredConceptMapArgs – say RequiredConceptMap.
  - if found, add RequiredConceptMap to RequiredConceptMaps
  - else if RequiredConcept is implicit, or to be treated as implicit (e.g. ParentMaps),
    - Generate the map – RequiredConceptMap.
    - if success, add to RequiredConceptMaps.
    - else, FAIL!
ConceptClang: Features Review

1. Features Implemented and Tested
   - Concept definitions (explicit)
   - Concept maps: definitions and instantiation.
   - Associated functions
   - Concept coverage and lookup
   - Concept refinement
   - Associated requirements
   - *late_check
   - Implicit concepts
   - *Explicit refinement
   - Constrained templates: constraints-check
   - Concept ids as qualified name

2. Features Implemented, but Probably Buggy
   - Scoped concepts
   - Associated function templates
   - Concept map templates
   - Associated types

... And We are Done (for now)! =D
ConceptClang: Features Review

1 Features Implemented and Tested
   - Concept definitions (explicit)
   - Concept maps: definitions and instantiation.
   - Associated functions
   - Concept coverage and lookup
   - Concept refinement
   - Associated requirements
   - *late_check
   - Implicit concepts
   - *Explicit refinement
   - Constrained templates: constraints-check
   - Concept ids as qualified name

2 Features Implemented, but Probably Buggy
   - Scoped concepts
   - Associated function templates
   - Concept map templates
   - Associated types

... And We are Done (for now)! =D
Use-Case Examples

1. Prototype Released: Alpha mode.
   - [http://zalewski.indefero.net/p/clang/](http://zalewski.indefero.net/p/clang/)
   - Download
   - Run Tests
   - Play!

2. Future Plans
   - Mini-BGL
   - stdlib
   - Others ???
Thank You!