C# language verification

Translation of C#-light language's syntactic constructions into USL expressions as the stage of C#-light programs verification.

Ivan S. Zapreev, 2003
Institute of Informatics Systems
E-mail: cerndan@mail.ru

Project participants

Leader: Professor V.A. Nepomniaschy

Research assistants:

- Dr. I.S. Anyreev
- A.V. Promsky

PhD. students:

- I.S. Zapreev
- I.V. Dybranovsky

Institute of Informatics Systems,
Laboratory of Theory of Programming,
Lavrentieva 6, Novosibirsk, Russia, 630090

C# language

- C# language was released by Microsoft in July 2000, as part of its .NET Framework initiative.
- C# is a simple, modern, general-purpose, object-oriented programming language.
- C# has much in common with C++ and Java
- The language, and implementations thereof support:
  - strong type checking,
  - array bounds checking,
  - detection of attempts to use uninitialized variables,
  - automatic garbage collection.

Hoare's axiomatic method

- In 1969 Hoare introduced an axiomatic method of proving programs correctness.
- The basic formulas of Hoare's logic are constructs of the form $\{P\}S\{Q\}$
  - S is a program
  - P, Q are assertions
- The meaning of the construct $\{P\}S\{Q\}$ is as follows: whenever (*precondition*) P holds *before* the execution of S and S terminates, then (*postcondition*) Q holds after the execution of S.
- Semantics of every simple statement is defined by axioms scheme and every composite statement is described by proof rules scheme.

A two layered verification

1. Definition of a "good" subset of C# language called C#-light and creation of its operational semantics;
2. Definition of C#-kernel language which is a subset of C#-light. C#-kernel have the same operational semantics as C#-light but it is possible to build a simple axiomatic semantics for it;
3. Definition of transformations from C#-light syntactical constructs to C#-kernel syntactical constructs and proving their correctness depending on the operational semantics;
4. Proving of consistency of the C#-kernel's axiomatic semantics regarding the C#-light's operational semantics.

C#-light programs verification

1. Take an annotated program written in the C#-light;
2. Translate it into C#-kernel;
3. Create verification conditions with lazy computations depending on the C#-kernel's axiomatic semantics;
4. Refine verification conditions. I.e. resolve lazy computations via partial program interpretation;
5. Prove verification conditions and analyse results.

Unified Semantic Language (USL)

- USL has been created as the result of generalization of different approaches to programming languages' formal semantics specification.
- The main constructs of USL are names and expressions:
  - **Names** are given by symbol sequences. There is a partial function (called a state) that maps names to expressions.
  - **Expressions** are built from names by operations. Any expression has a value and can change the state. Expressions are classified as: *atomic*, *mathematical*, *structural*, *semantic*, *logical*, *expressions with side effects* and *comments*.

USL Expressions

- **Atomic expressions** are *names*;
- **Semantic expressions** serve to change the expression semantics. They are built by the operations & and *;
- **Mathematical expressions** are built by:
  - *tuple [ ],*
  - *set { },*
  - *map < >,*
  - the *plus* and *minus* operations;
- **Structural expressions** serve to give the order of sub expressions evaluation and to structure expressions. They are:
  - *empty expression,*
  - *parenthesized expression,*
  - *sequential grouping,*
  - *sequential execution,*
  - *substitution,*
  - *conditional expression,*
  - *action*;

USL Expressions

- **Logical expressions** are built by propositional connectives:
  - *and*, *or*, *xor*,
  - *imply*,
  - *iff*,
  - *not*,
  - *quantifiers exist*, *exists!*, *forall*,
  - *eq*;
- **Comments** are built by the operation *comment*;
- **Expressions with side effects** serve to change a state. They are built by the following operations:
  - assignment *assign*,
  - application *( )*,
  - the operation *new* and *delete* (changing a state domain),
  - the operation *return* (returning the action value).

C#-light language

- The C#-light is a subset of the C# language. It allows writing sequential programs and contains all C# language constructs except:
  - *threads*;
  - *attributes*;
  - *unsafe* code;
  - *destructors*;
  - *lock* and *resource* statements;
  - *checked* and *unchecked* constructs.

C#-light's operational semantics

- The C#-light language operational semantics definition requires an abstract machine specification that in its turn demands to determine the Abstract Machine (AM) states and behavior. Each state is defined in terms of a language entity and a language object.
- A language entity (*variable*, *statement*, *class definition*, ...) is defined by its type and a set of attributes. For instance, a "class definition" entity of the C#-light has a *class-declaration* type and *attribute-sections*, *modifiers*, *name*, *inherited-class*, *implemented-interfaces* and *members* attributes.
- A language object is an instance of a language entity (a concrete class definition, variable, …). A value of a language object is represented as a USL expression of the following kind:

$$<[name\text{-}type, t], [a_1, e_1], \ldots, [a_n, e_n]>$$

where t is a language entity type, $a_1, \ldots, a_n$ are attribute names, $e_1, \ldots, e_n$ are corresponding attribute values.

C#-light's operational semantics

- A state of AM is a USL state (names present language objects and the state defines the values of these objects).
- The behavior of AM is specified by a USL action.

  For the C#-light's AM, action is called C#-object-evaluation it has the following form:

  set(&C#-object-evaluation,action(x,if(is-C#-object(x),
  *concatenation(x(&name-type), &-action)(&x), &syntactical-error)))

  – Check whether the input name x is a C#-light object.
  – If not, the result is the syntactical-error expression.
  – Otherwise, the action that evaluates C#-light objects of this type is executed.
- AM "understands" only the language of states that is why we have to translate programs into states.

USL Expression example

The local variable declaration with initialization:

int i = 0;

Is translated into the following USL expression:

```
<[name-type, local-variable-declaration],
  [type,int],
  [declarators,
   [<[name-type, local-variable-declarator],
       [name, i],
       [initializer,0]>]]>]
```

The C#-kernel language

- C#-kernel does not contain namespaces and using-directives
- All C#-light statements are eliminated, except:
  - local variable and constant declaration-statement;
  - expression-statement;
  - block;
  - labeled-statement;
  - if-statement;
  - goto-statement.
- The following operators are not allowed in C#-kernel expressions:
  - logical operators || and &&,
  - conditional operator ?:,
  - all compound assignment operators (except when the left operand of the += or -= operator is a normalized expression that is classified as an event access).
- A function member is allowed to be invoked only in its normal form.

C#-kernel's axiomatic semantics

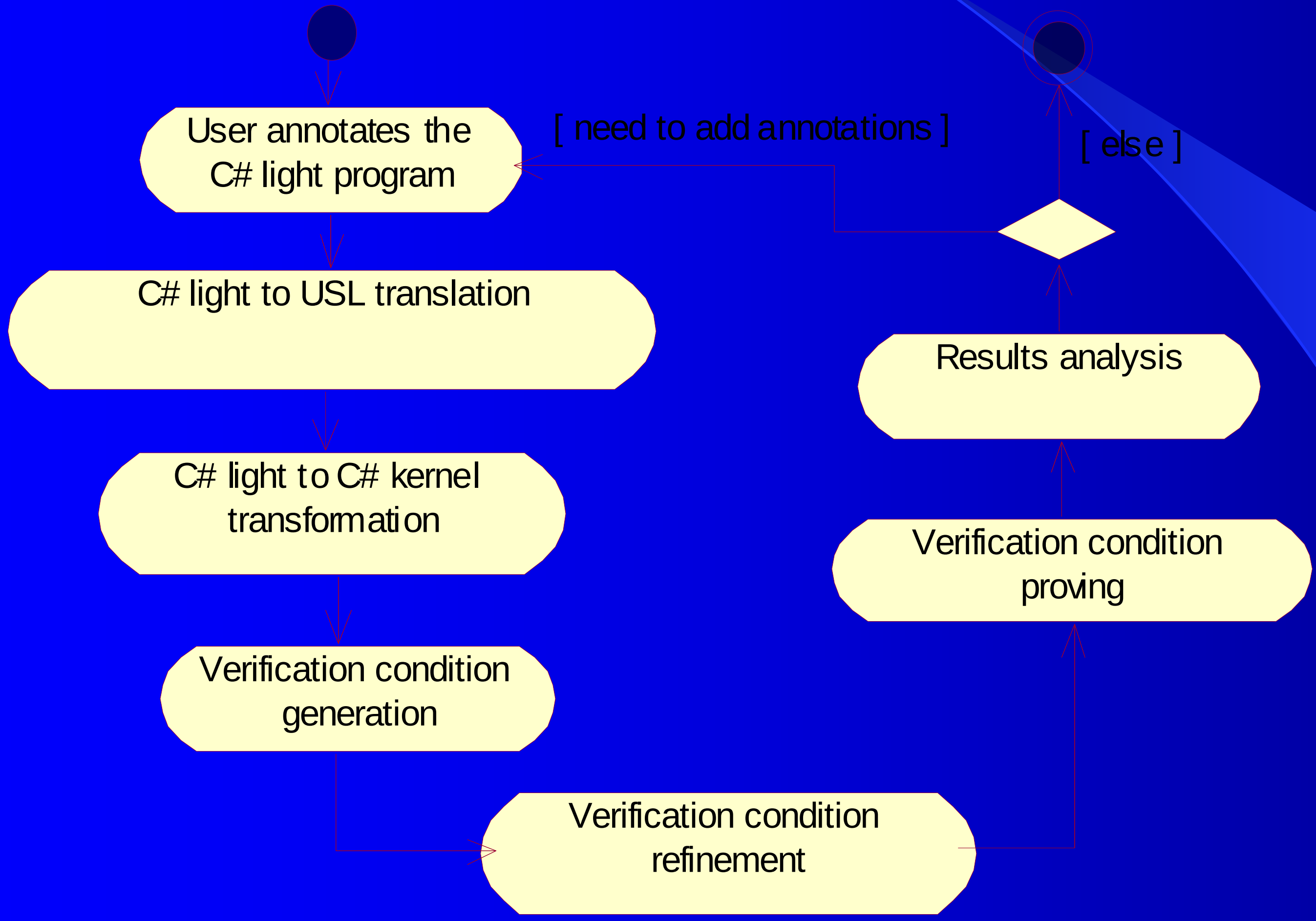
- In 1969 Hoare introduced an axiomatic method of proving programs correct.
- Real execution is replaced by symbolic manipulations over logical formulas. Not every construct can be formalized correctly.
- C#-light program is translated into C#-kernel and "bad" constructs are replaced by equivalent C#-kernel fragments.
- All constructs of C#-kernel can be axiomatically formalized.

C#-light programs verification system

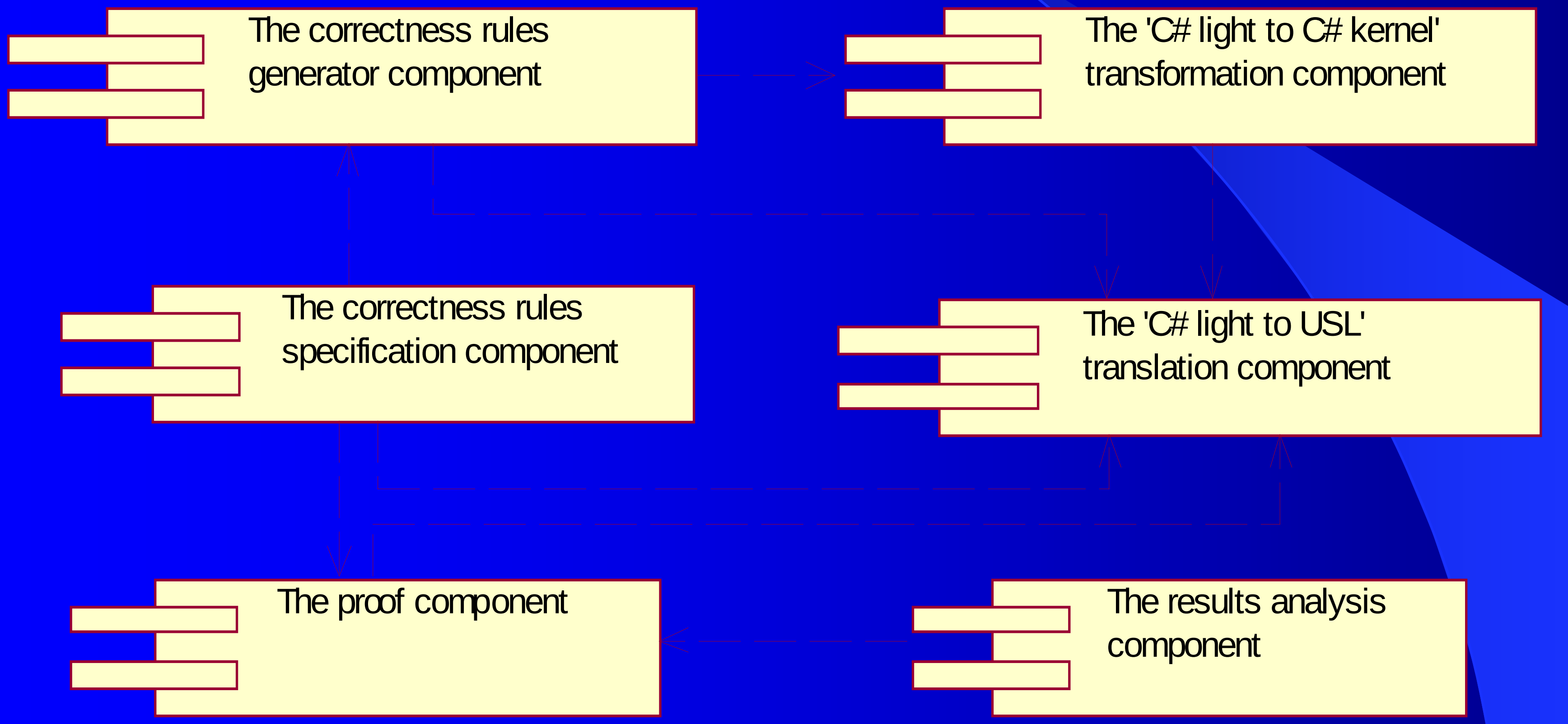
The developing C#-light programs verification system consists of the following components:

- The "C#-light to USL" translation component
- The "C#-light to C#-kernel" transformation component
- The verification condition generator
- The verification condition qualifier
- The prover
- The result analyzer

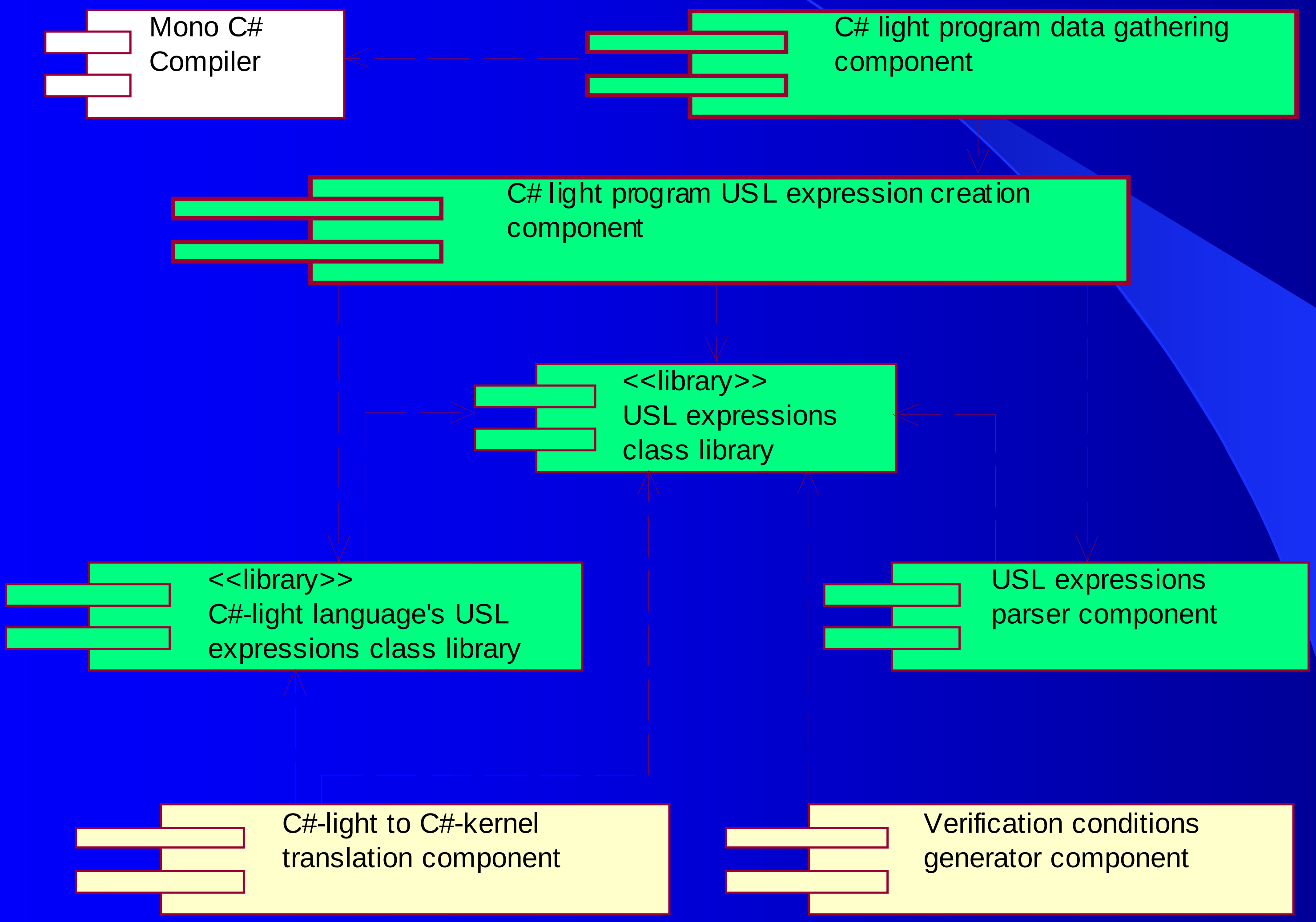
Verification process



The Verification system components



The C#-light to USL translation component



Gather a C#-light program's data

How to get information about the C#-light program:

- Use a third party C# parser,
- Create a special C#-light parser.

Currently we use an open source C# compiler called Mono distributed by the Ximian company (http://www.go-mono.org/).

We are going to use Common Compiler Infrastructure in the nearest future instead.

C#-light to USL, the future.

- Currently the main goal of the C#-light to USL translation is to separate from the C#-light program's data provider and to store this data in the useful and universal internal representation (IR).
- In the future we propose to create a USL interpreter which will take:
  - A USL representation of the program,
  - A corresponding programming language's operational semantics (C# in our case) defined via USL;

  and thus will be able to interpret it.

  This can be widely used in:
  - Testing,
  - Static analysis,
  - Debuggers implementations,
  - Runtime verification;

C#-light to USL, the main results

The following results were gathered while development of the C#-light to USL translator:

- A common approach to translation of C#-light programs into USL expressions has been developed;
- A USL classes library has been developed;
- A USL expression parser has been developed;
- A prototype of the “C#-light to USL” translation component has been developed on the bases of an open source C# compiler (Mono, Ximian).