Compiler Construction Using LOTOS NT

Hubert Garavel, Frédéric Lang, Radu Mateescu

INRIA Rhône-Alpes / VASY

655, avenue de l'Europe

F-38330 Montbonnot Saint Martin

France

Formal specification languages

- **LOTOS** [International Standard ISO 8807]
  - Communicating asynchronous processes
  - Abstract Data Types (equational programming)
  - Compilers: CAESAR [process part], CAESAR.ADT [data part]
- **E-LOTOS** [International Standard ISO 15437]
  - Enhancements to LOTOS (work between 1993 and 2001)
  - Timed communicating processes
  - Functional data types
  - Modules and interfaces
- **LOTOS NT** [INRIA/VASY]
  - Dialect of E-LOTOS
  - Since 1998: TRAIAN, a compiler for LOTOS NT data part

Overview of LOTOS NT data part

- A first-order functional language with an imperative syntax
- Data types
  - Base types: bool, int, real, string, …
  - Constructive types used to define abstract trees
  - Particular cases: enum, records, lists, trees, etc.
  - Fixed size arrays (not implemented yet)
- Functions
  - Functions with in/out/in-out parameters
  - Variable assignments and **return** statement
  - Side effects forbidden
  - Static analysis (typing, variable initialization, …)
  - Standard control structures (**if-then-else**, **while**, etc.)
  - Pattern matching (**case**)
  - Exception handling
- Connection to external C types and functions

The TRAIAN compiler

Generates C code for the LOTOS NT data part

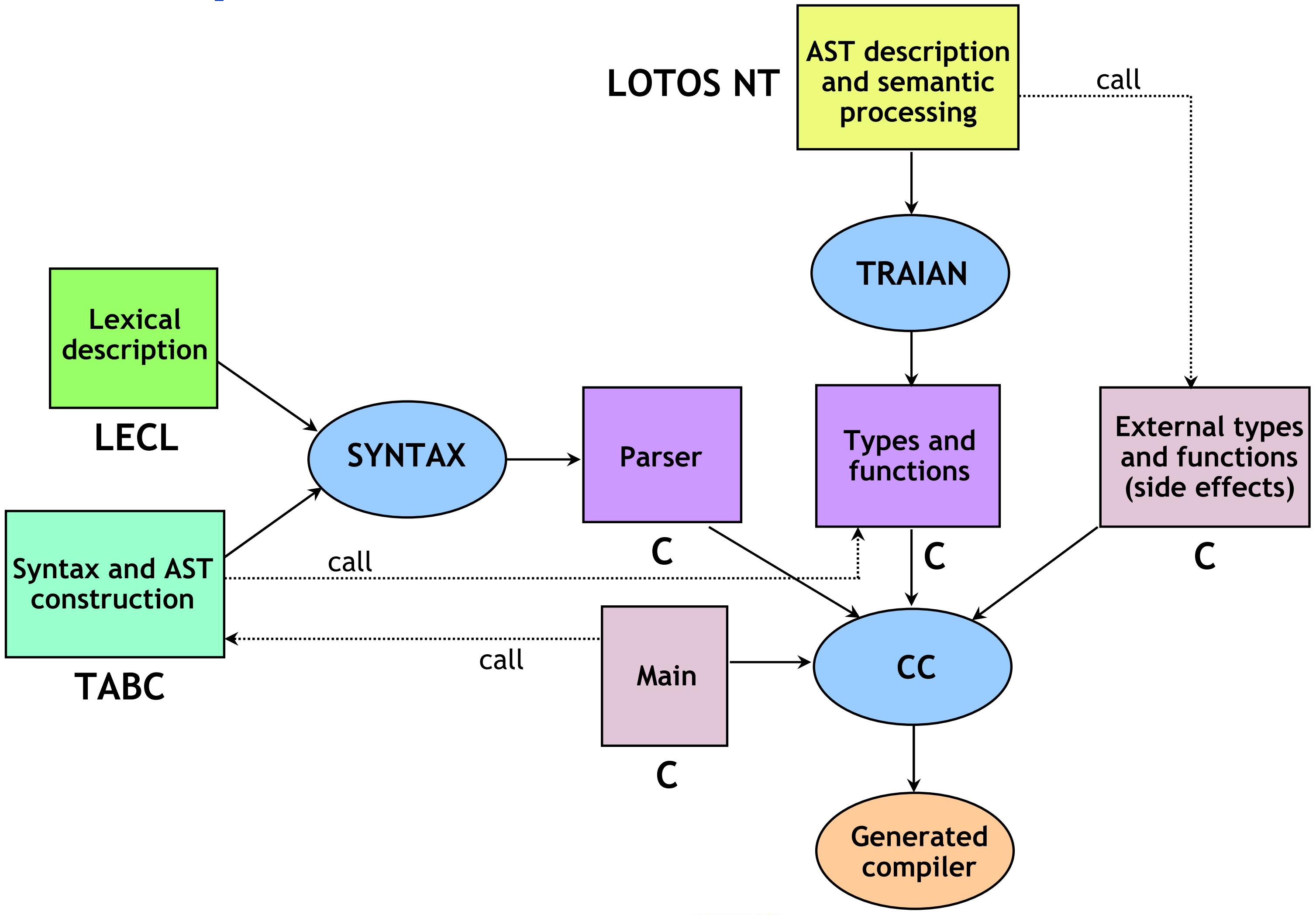
- TRAIAN 1.0 released in 1998
- TRAIAN 2.3 (April 2003)
  - More than 55 000 lines of SYNTAX + FNC2
  - Optimizations to reduce data space consumption: pointer minimization, particular types, etc.
  - Benefit from experience on CAESAR.ADT
- Free download: 71 sites in 2002

Compiler construction using TRAIAN

- Compiler construction technology based on
  - The SYNTAX parser generator (INRIA)
  - LOTOS NT and the TRAIAN compiler

- Illustration: Statements of a simple procedural language named simproc

The SYNTAX + TRAIAN compiler construction technology



Abstract tree definition
(excerpt of simproc.lnt – LOTOS NT)

```
type STMT is !implementedby "C_TYPE_STMT"
  ASSIGN (V : VARIABLE, E : EXPR)
    !implementedby "C_ASSIGN",
  --
  CALL (PROCID : ID, ACTUALS : PARAMETER_LIST)
    !implementedby "C_CALL",
  --
  IF_THEN_ELSE
    (
      E : EXPR,
      S_THEN : STMTS,
      S_ELSE : STMTS
    ) !implementedby "C_IF_THEN_ELSE",
  --
  …
end type
```

Lexical description
(simproc.lecl - LECL)

```
Classes
   SPACE           = SP + HT + NL + FF ;

Tokens
   Comments         = -{ SPACE | "%" "%" {^EOL}* EOL }+ ;

   %ID            = LETTER {["_"] (LETTER | DIGIT)}* ;

   %INT            = {DIGIT}+ ;
```

Syntax description
(excerpt of simproc.tabc – TABLES C)

```
* Attribute declarations
$TABC_STMT (<BNF_STMT>) : C_TYPE_STMT ;
$T ABC_VARIABLE (<BNF_VAR>) : C_TYPE_VARIABLE ;
$TABC_EXPR (<BNF_EXPR>) : C_TYPE_EXPR ;
…
* BNF rules and attribute definitions

<BNF_STMT> = <BNF_VAR> ":=" <BNF_EXPR> ;
$TABC_STMT (<BNF_STMT>)
$TABC_STMT (<BNF_STMT>) =
    C_ASSIGN ($TABC_VARIABLE (<BNF_VAR>), $TABC_EXPR (<BNF_EXPR>));

<BNF_VAR> = %ID;
$TABC_VARIABLE (<BNF_VAR>)
$TABC_VARIABLE (<BNF_VAR>) = C_VAR ($pste ("%ID"));
…
```

AST traversals
(excerpt of simproc.lnt – LOTOS NT)

```
function CHECK_STMT (INSTR : STMT, SYMBOLS : S_TABLE) : BOOL is
    case INSTR is var … in
      ASSIGN (VAR1, EXP1) ->
        var … in
            VAR_TYPE := CHECK_VAR (VAR1, SYMBOLS);
            EXPR_TYPE := CHECK_EXPR (EXP1, SYMBOLS);
            CORRECT := (VAR_TYPE == EXPR_TYPE) and (EXPR_TYPE != TYPE_ERROR);
            if not CORRECT then
                    eval PRINT_ERROR ("type mismatch")
            end if;
            return CORRECT
        end var
    | CALL (PROCID, ACTUALS) -> …
    | IF_THEN_ELSE (EXP1, INSTS1, INSTS2) -> …
    end case
end function
```

External types

Excerpt of simproc.lnt

```
type SYMBOL_TABLE is
   !external !implementedby "C_SYMTAB"
end type
```

Excerpt of simproc.t

```
typedef struct  {
            …
} C_SYMTAB [MAX_ENTRIES];
```

External functions

Excerpt of simproc.lnt

```
function PRINT_ERROR (S : STRING) is
  !external !implementedby "C_EXT_PRINT_ERROR"
end function
```

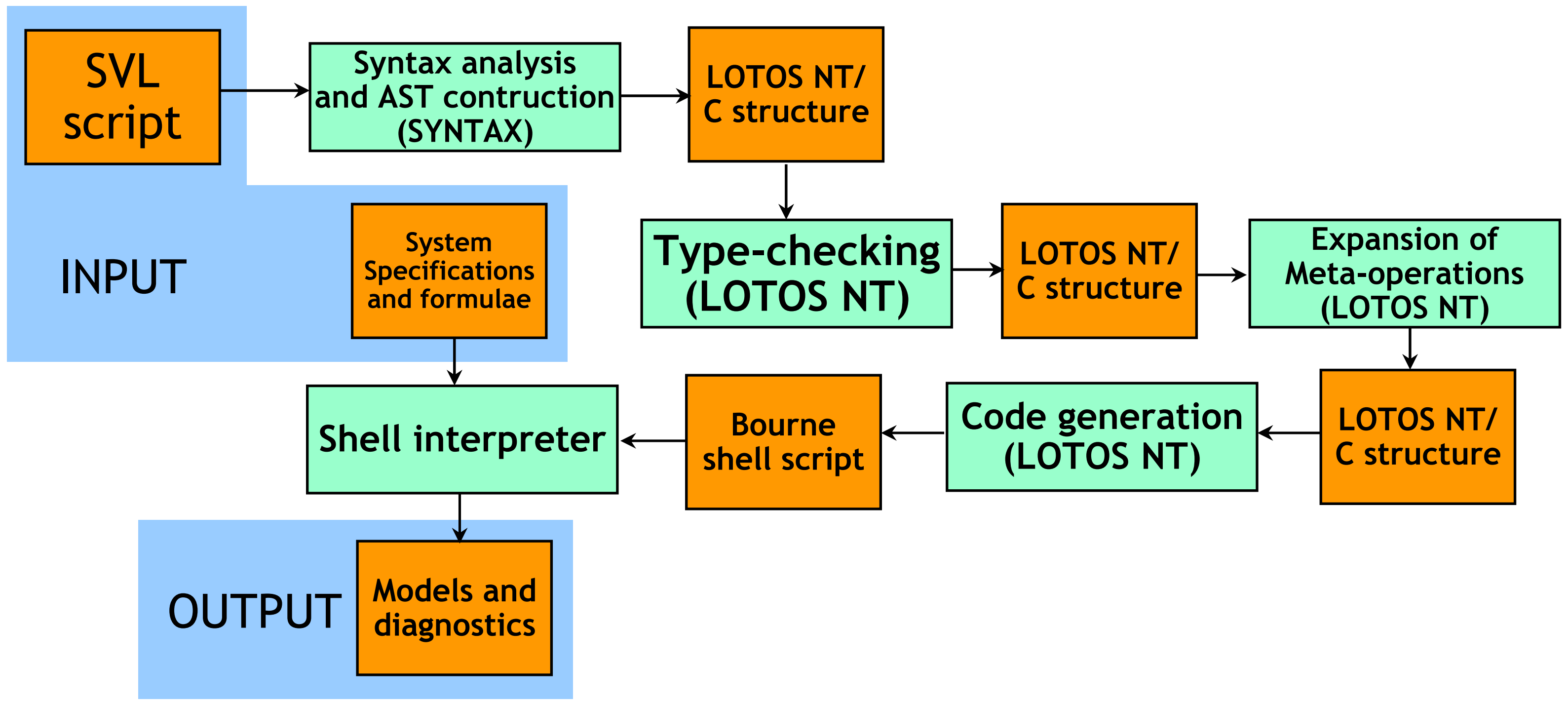
Excerpt of simproc.f

```
void C_EXT_PRINT_ERROR (ERROR_MSG)
     ADT_STRING ERROR_MSG;
{
  ADT_PRINT_STRING (stdout, "error : ");
  ADT_PRINT_STRING (stdout, ERROR_MSG) ;
  ADT_PRINT_STRING (stdout, "\n");
}
```

Compilers already developed using LOTOS NT

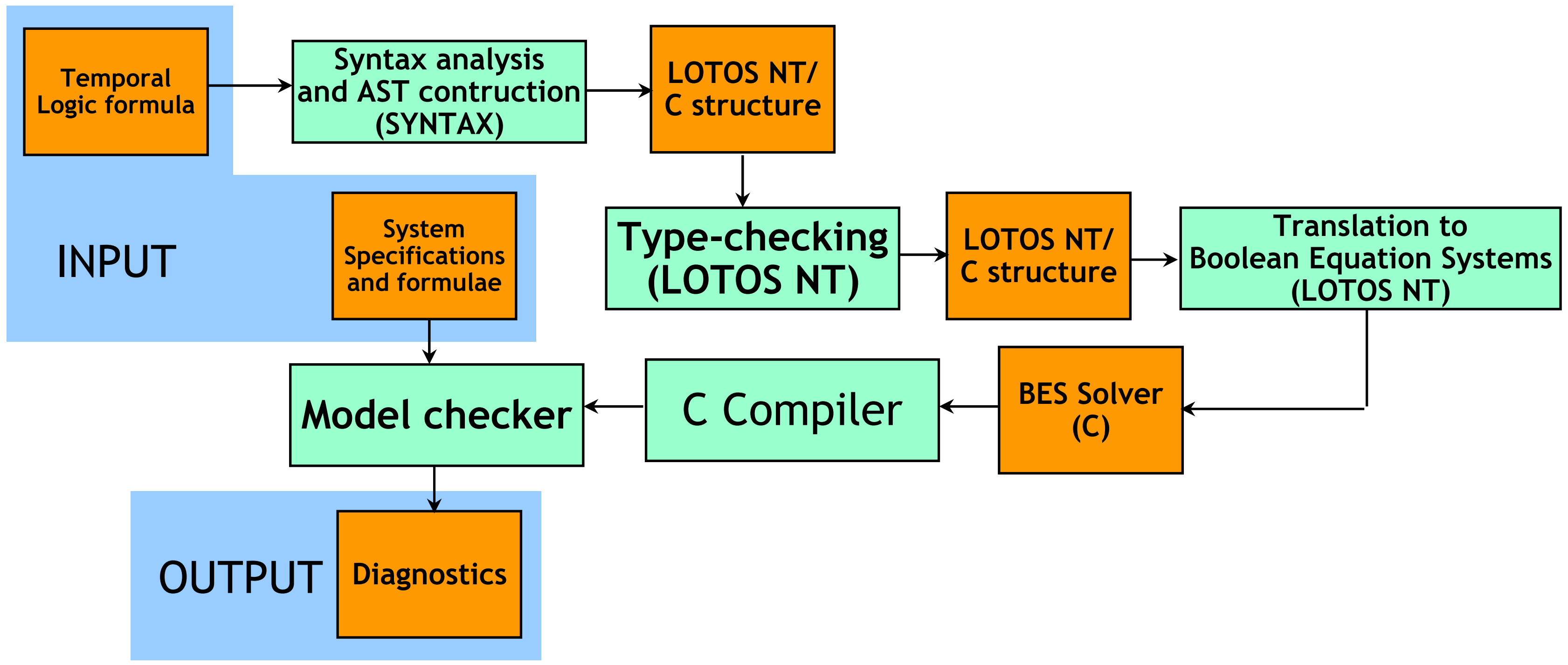
The SVL 2.0 compiler

Scripting language dedicated to verification



- SYNTAX: 1 250 lines
  LOTOS NT: 2 940 lines ⇒ generated C: 12 400 lines
  Hand-written C: 370 lines
- Distributed within CADP since July 2001

The Evaluator 4.0 model checker

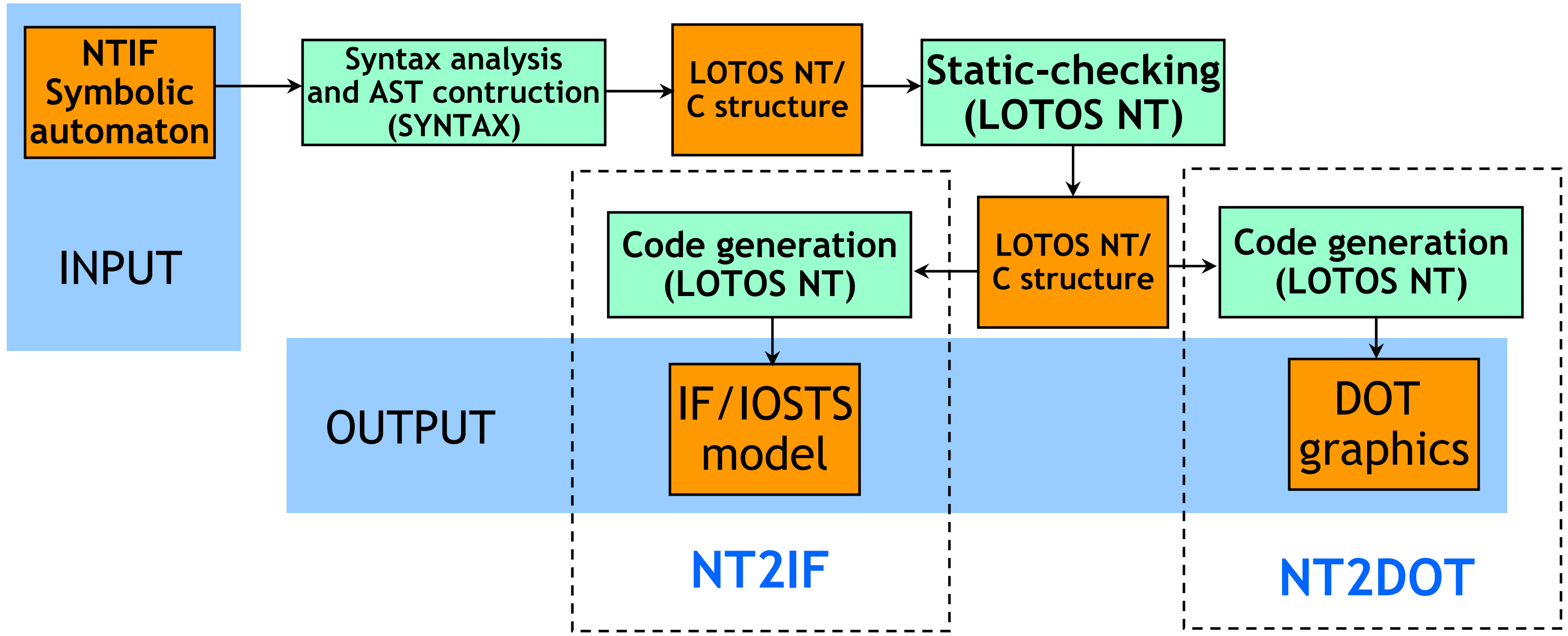


• SYNTAX:	3 600 lines	
LOTOS NT:	11 500 lines	⇒ generated C: 45 000 lignes
Hand-written C:	3 900 lines	

- Distributed in next release of CADP

The NTIF tools

Symbolic automata processing



NTIF Symbolic automaton
INPUT
Syntax analysis and AST contruction (SYNTAX)
LOTOS NT/ C structure
Static-checking (LOTOS NT)
LOTOS NT/ C structure
Code generation (LOTOS NT)
Code generation (LOTOS NT)
OUTPUT
IF/IOSTS model
DOT graphics
NT2IF
NT2DOT

• SYNTAX:	1 620 lines	
LOTOS NT:	3 620 lines	⇒ generated C: 20 600 lignes
Hand-written C:	1 200 lines	

Strengths of the technology

- Fast development
  - The technology is simple and easy to learn
  - SYNTAX flexibility for parser generation: accepts a large class of BNF grammars, powerful error recovery
- Maintainable and robust code
  - Readable LOTOS NT code
  - TRAIAN static checks: strong typing, case exhaustivity, uninitialized variables, uncaught exceptions
  - Direct pointer manipulations avoided
  - Efficient generated code

Strengths of the technology (continued)

- Portability
  - Tools available on Solaris, Linux, and Windows
  - Standard C code generated
  - Straightforward interface with C
- Life time
  - SYNTAX is stable and mature
  - LOTOS NT / TRAIAN are stable and actively supported

Conclusions

- A simple and working solution…
- LOTOS NT: A formal specification language well-suited to implementing compilers
- Future:
  - New tools will be developed using this technology
  - Bootstrap: TRAIAN 3.0 written in LOTOS NT
- TRAIAN is freely available at

http://www.inrialpes.fr/vasy/traian