PLUSS

A LANGUAGE FOR STRUCTURED SPECIFICATIONS

ASSPEGIQUE

AN ENVIRONMENT FOR FORMAL SPECIFICATION

M-C. Gaudel
M. Bidoit, B. Biebow, C. Choppy,
S. Kaplan, A. Mauboussin, F. Voisin
YET ANOTHER SPECIFICATION LANGUAGE?

PLUSS is the result of several experiments in writing large specifications
- using algebraic data types
- in collaboration with people from industry

STARTING POINTS
- ASL primitives (Wirsing & Sanella)
- Some software tools
  parser (CIGALE), symbolic evaluator (SPECTRAL)...
  - PERQ workstations

CASE STUDIES
- Recorders management in E10S
  (a telephone switching system)
- CSE Electronic Satellite Concentrator
  (together with Petri Nets)
  - PASCAL into P-Code Compiler
  - ADA Data Types
  - Level 2 & 3 protocols in CCITT no 7

CURRENT RESEARCHES
- PLUSS: Specification Language
- ASSPEGIQUE: Integrated Specification Environment
  - MAIA: Lisp-Prolog Workstation
LARGE SOFTWARE PROJECTS
AND FORMAL SPECIFICATIONS

OPEN PROBLEMS (some of)

• STRUCTURING & MODULARIZING
  SPECIFICATIONS

• COEXISTENCE OF VARIOUS
  FORMALISMS (f.i. Algebraic spec. +
  Petri Nets)

• IMPACT ON OTHER STEPS OF THE
  DEVELOPMENT PROCESS (f.i. Testing)

Hierarchical Specifications
are easy to understand.
They are difficult to
design.

When designing a specification,
you want to modify the
class of models.

When using a specification,
you must not modify the
class of models.
REMARKS

- A Specification describes a class of implementations (i.e. Algebras)

- When you read a specification you assume that only finitely generated algebras are considered.

- When you write a specification you assume that not finitely generated algebras are considered.

A TOUR OF PLUSS
in one slide

BASED ON ASL PRIMITIVES

specif use
proc } parameterized specifications
param
draft enrich union

renaming

visibility rules

observe

signatures
specif TABLE =
  use ELEMENT

  sort Table
  operations
    empty-table: -> Table;
    insert _ into _: Element x Table -> Table;
    _ is in _: Element x Table -> Bool;
    remove _ from _: Element x Table -> Table;
  gen empty-table, insert _ into _

  axioms
    % definition of is-in %
    e is-in empty-table = false;
    e is-in (insert e' into t) = (e is e') or (e is-in t);
    % definition of remove %
    remove e from empty-table = empty-table;
    e is e' = false => remove e from (insert e' into t) =
      insert e' into (remove e from t);
    e is e' = true => remove e from (insert e' into t) =
      remove e from t

  where
    e, e': Element;
    t: Table
end TABLE

end TABLE

The TABLE specification

Example of term:
  insert e into t

Example of ground term:
  remove a from insert b into insert a into empty-table

specif COMPARABLE-TABLES =
  use TABLE

  operation
    _ is included-in _: Table x Table -> Bool
    % tests inclusion of tables %

  axioms
    empty-table is included-in t = true;
    (insert e into t) is included-in t' =
      (e is-in t') and (t is included-in t')

  where
    t, t': Table;
    e: Element
end COMPARABLE-TABLES

Example of use of the TABLE specification
proc PTABLE(ITEM) =
    sort Ptable
    operations
        empty: -> Ptable;
        add-entry : Item × Ptable -> Ptable;
        suppress : Item × Ptable -> Ptable;
        _belongs-to_: Item × Ptable -> Bool;
    gen empty, add-entry
    axioms
        i belongs-to empty = false;
        i belongs-to add-entry(i', t) = (i eq i')
            or (i belongs-to t);
        suppress(i, empty) = empty;
        i eq i' => suppress(i, add-entry(i', t)) =
            suppress(i, t);
        i eq i' => suppress(i, add-entry(i', t)) =
            add-entry(i', suppress(i, t))
    where
        i, i' : Item;
        t : Ptable
end PTABLE

Specification of Parameterized Tables

param ITEM =
    use BOOL
    sort Item
    operation
        _eq _: Item × Item -> Bool
    axioms
        % eq is an equivalence %
        i eq i = true;
        i eq j = j eq i;
        i eq j = true & j eq k = true => i eq k = true
    where
        i, j, k : Item
end ITEM

Specification of the Properties
Required for the Arguments

specif TABLE =
    PTABLE(ELEMENT)
    renaming Ptable into Table,
        empty into empty-table,
        add-entry into insert _into_,
        suppress into remove _from_,
        _belongs-to_ into _is-in_
end TABLE

Making the TABLE Specification
from the Parameterized One
specif SPEC export s, op, SPEC1 =
use SPEC1, SPEC2
sorts s, s'
operations
op : ... ;
axioms ...
end SPEC

specif SPEC forget s', SPEC2 =
use SPEC1, SPEC2
sorts s, s'
operations
op : ... ;
axioms ...
end SPEC

specif RESTRICTED-TABLE forget insert _into_,
remove_from_
use TABLE
end RESTRICTED-TABLE

Visibility Clauses

specif CONTINUUM =
use MESSAGE, INTEGER
sort Board, Erroneous-Message
operations
new-board : Board;
add-new-message : Message • Board -> Board;
erase-message : Int • Board -> Board;
read : Int • Board -> Message U Erroneous-Message;
size : Board -> Int;
%the following operations correspond to error messages%
erased-message -> Erroneous-Message;
no-such-message : -> Erroneous-Message;
gen new-board, add-new-message, erase-message,
erased-message, no-such-message
axioms
size(new-board) = 0;
size(add-new-message(m, b)) = size(b) + 1;
size(erase-message(n, b)) = size(b);
read(n, new-board) = no-such-message ;
- n is size(b) = true =>
read(n, add-new-message(m, b)) = m ;
- n is size(b) = false =>
read(n, add-new-message(m, b)) = read(n, b) ;
- n is n' = true =>
read(n, erase(n', b)) = erased-message ;
- n is n' = false =>
read(n, erase(n', b)) = read(n, b) ;
where
b : Board ;
m, m' : Message ;
n : Integer ;
end CONTINUUM
specif USER-CONTINUUM
export MESSAGE, INTEGER, Board, Erroneous-Message, add-new-message, read, size, erased-message, no-such-message = CONTINUUM
end USER-CONTINUUM

specif CHAIRMAN-CONTINUUM =
use CONTINUUM, USER, PTABLE(USER)
% USER defines the attributes of users.
Besides this specif uses tables of users which are renamed as users-lists.
renaming Ptable into Users-list; ...
end CHAIRMAN-CONTINUUM
draft FILE-T =
use NAME, TEXT

sort File
operations
name _: File -> Name ;
content _: File -> Text ;
<_, _> : Name x Text -> File

axioms
name <n, t> = n ;
content <n, t> = t ;
where
n : Name ;
t : Text
end FILE-T

File of Text : Draft

draft FILE-B =
use NAME, BINCODE

sort File
operations
name _: File -> Name ;
content _: File -> Bincode ;
<_, _> : Name x Bincode -> File

axioms
name <n, c> = n ;
content <n, c> = c ;
where
n : Name ;
c : Bincode
end FILE-B

File of Binary Code : Draft
draft FILE-E =
use NAME
sort File
operations
create : Name -> File
name _ : File -> Name
axioms
name create(n) = n
where
n : Name
end FILE

Empty File : Draft

draft FILE =
enrich FILE-E, FILE-T, FILE-B by

operation
file _ : File -> {empty, englishText, executable}

axioms
file create(n) = empty :
file <n, t> = t :
file <n, c> = c
where
n : Name ; t : Text ; c : Bincode
end FILE

specif FILE from FILE =
gen < _, _>, create
% possibly some new operations can be added here %
end FILE

Making the FILE specification from Drafts

end FOREST

Specification of Directories

n : Name ; F : Forest
where
file <n, F> = n :
name-list-of (φ) = directory;
name <n, F> = n :
file <n, F> = F :
name-list-of (φ) = directory ;
name-list-of (φ) = A :
name-list-of (φ) = (name φ), name-list-of (φ) ;
draft SYSTEM =
use PATH
enrich FOREST by

sort System
operations
  root : -> System;
  mkfile : System x Path x File -> System;
  mkdir : System x Path x Name -> System;
_: System -> Directory;

% This is a coercion: a system is a directory and
all the operations on directories can be applied
 to systems. %
ls : System x Path -> Namelist;
......

axioms
  p is p' = true => ls(mkfile(s,p,f),p') = (name f)·ls(s,p');
  p is p' = false => ls(mkfile(s,p,f),p') = ls(s,p');
  p is p' = true => ls(mkdir(s,p,n),p') = n·ls(s,p');
  p is p' = false => ls(mkdir(s,p,n),p') = ls(s,p');
......

where
  p, p' : Path ; s : System ;
  f : File ; n : Name
end SYSTEM

Specification of a File System: Draft

specif SYSTEM from SYSTEM =
gen root, mkfile, mkdir for System ;
ϕ, _♣_ for Forest;
< _, _> for Directory;

% operations such that rm or rmdir
 can be specified here. %

end SYSTEM

Making the FILE SYSTEM specification from the Draft
The design of ASSPEGIQUE

Two main motivations:

* Tools to make experiments on various theories, to test them.

* Tools to write and handle large specifications.
Ease of use, flexibility, user-friendly interfaces

* on-line documentation and help
* uniform multi-window interfaces
* WINNIE
* specific windows for the display of error messages, information...
* graphic interfaces
  * pop-up menus, pointing device,
  * graphic representation of signatures, terms, proof steps, relations between specifications...
  * mixed syntax is allowed for operation symbols

Identification of the tools required

* Hierarchical library management tool

* CIGALE: a tool for interactive and incremental grammar construction and expressions parsing.

Kernel specification environment:

Editor → Compiler → Symbolic Env.

2nd Level specification environment:

* Modification and reuse tools
* Interactive debugger
* Theorem proving tools...
SPEC: DICTIONARY
  SORTED-ARRAY (SORTED-THING => WORD )
END DICTIONARY.

SPEC: COMPARABLE-WORDS
  WITH WORD
END WORD.

PROC: SORTED-ARRAY (SORTED-THING)
  WITH INTEGER
  SEQUENCE (X => CHAR)
END SORTED-ARRAY.

PAR: SORTED-THING
  WITH BOOL
END SORTED-THING.

SPEC: INTEGER
  WITH BOOL
END INTEGER.

SPEC: CHAR
  WITH BOOL
END CHAR.

SPEC: BOOL
END BOOL.

PROC: SEQUENCE (X)
  WITH BOOL
END SEQUENCE.

PAR: X
END X.

PAR: SORTED-THING
WITH BOOL
END SORTED-THING.

SPEC: WORD
  SEQUENCE (X => CHAR)
END WORD.

PROC: SORTED-ARRAY (SORTED-THING)
  WITH INTEGER
END SORTED-ARRAY.

Note: There are two layers of syntax:
  - the specification language 'SPEC'
  - the term level in the axioms 'CIGALE'

- CIGALE -

  • Syntax of the operators must be free
  • enhance flexibility, legibility, correctness
  • "mixfix" syntax à la Ober, with ""-"" to design the position of an operand
    e.g.: push_out; data stack -> stack

  • Incremental construction of the
    language
  • use of specifications, debugging
    of specifications
  • interactively add/delete operators
CiGALE (cont.)

- Modularity
  - notation of a "current parsing environment"
  - interactively add/delete (sub-)languages.

- Coercion, overloading and ambiguities
  e.g.: 
  \[ - : \text{integer} \to \text{monomial} \]
  \[ - : \text{monomial} \to \text{polynomial} \]

or

push - out - : data stack \to stack
append - to - : data file \to file
x i \to data

push x onto empty
append x to empty

useful ... and even necessary for variables!

ambiguities \& "most natural analysis"
The specification compiler

* computes the grammar associated to an elementary specification:

\[ \text{gram}(\text{spec}) = \cup \text{gram} \text{(spec)} + \Sigma \text{spec} \text{, required by spec} \]

* slightly different on circles

* parses the axioms

* creates an internal representation and updates library information

2 compilation of a special function cannot occur unless required specifications are successfully checked.

The internal representation produced by the compiler is the form that will be directly used by the other tools: symbolic evaluation, theorem proving tools, ...

The debugger

is loaded when errors are detected during compilation

allows to interactively debug axioms
Term rewriting system (computes the canonical form of an expression w.r.t. the axioms).

- allows conditional axioms
takes parameterization into account.

Theorem proving tools:
- Generator induction in "fairly" specified
  equational theories
  (extension to conditional ones is
   investigated)
- will be used to verify correctness of
  - parameter passing
  - implementations

#### SPEC : SEQUENCE
WITH : BOOL
SORTS : Seq
OPERATIONS : Use INS to edit operations and sorts through the FCM
  empty : → Seq
  cons : → X → Seq → Seq
  car : → X → Seq → X
  cdr : → X → Seq → Seq
  append : → X → Seq → Seq
  less than : → X → Seq → Seq
  Value : t

VARIABLES :
  s1, s2 : Seq
  x : X

AXIOMS :
  car-1 : car cons x1 x = x
  cdr-1 : cdr cons x1 x = s1

(ASSEDIQUE(mod)> Value : t
CONCLUSION

Asspeigue is implemented in Franz Lisp under VAX/unix.

> still under development

> evaluation and collaboration with industrial people will provide a firm basis to further developments