

Sviluppo di Software Sicuro - S³
SPARK – Moduli

Corso di Laurea Magistrale in
Sicurezza Informatica: Infrastrutture e Applicazioni
Università di Pisa – Polo di La Spezia
C. Montangero
Anno accademico 2009/10

Sommario

- Preliminari: intervalli e array
- Oggetti e tipi astratti: Un trito esempio
- Moduli come oggetti
- Importazioni
- Moduli per tipi di dati astratti (ADT)

S3: SPARK - C.Montangero - Copyright 2010 2

S³ 2009/10 – SPARK – Moduli

INTERVALLI E ARRAY

S3: SPARK - C.Montangero - Copyright 2010 3

Sottotipi: intervalli

```
-- subtype Natural is Integer range 0 .. Integer'Last;
-- subtype Positive is Natural range 1 .. Natural'Last;

ThisStackSize : constant Integer := 100;
subtype PointerT is Natural
  range Natural'First .. ThisStackSize;
Pointer : PointerT;
```

- Definibili sui tipi discreti
- Sugli intervalli sono definiti gli attributi
 - First e Last, valori del tipo discreto da cui si parte
 - Range, da usare dove serve un intervallo (loop, aggregati, ...):
 - T'Range è come T'First .. T'Last

S3: SPARK - C.Montangero - Copyright 2010

4

Cicli for

```
iteration_scheme ::= ... | for loop_parameter_specification

loop_parameter_specification ::=
  defining_identifier in [reverse] discrete_subtype_definition

for J in Buffer'Range loop -- works even with a null range
  if Buffer(J) /= Space then
    Put(Buffer(J));
  end if;
end loop;
```

- J è una costante (diversa ad ogni ciclo)
- Il tipo è quello dell'intervallo

S3: SPARK - C.Montangero - Copyright 2010

5

Dichiarazione di tipi array

```
type IntVectorT is array (Integer range <>) of Integer;
subtype StackIndexRange is Positive
  range 1 .. ThisStackSize;
subtype IntStackT is IntVectorT(StackIndexRange);
```

- Aggregati: espressioni di tipo array
 - (StackIndexRange => 0)
 - IntStackT'(0, 0, ... , 0) - StackIndexRange volte
 - IntStackT'(1 => 0, others => 16#0000_0000#)
- IntVectorT è svincolato, IntStackT vincolato
- Un oggetto di tipo array svincolato accetta valori su intervalli diversi
 - ridefinisce i propri attributi First, Last, Range

S3: SPARK - C.Montangero - Copyright 2010

6

Tipi array vincolati e no

```
type IntVectorT is array (Integer range <>) of Integer;
subtype StackIndexRange is Positive range 1 .. ThisStackSize;
subtype IntStackT is IntVectorT(StackIndexRange);
```

- IntVectorT è *svincolato*
 - gli attributi First, Last, Range *non* sono definiti
- IntStackT è *vincolato*
 - gli attributi First, Last, Range sono definiti, in base al subtype
- Gli oggetti di tipo array hanno gli stessi attributi
 - per tipi array svincolati definiti dinamicamente
 - e possono cambiare durante l'esecuzione
 - per esempio se son parametri di un sottoprogramma
 - per quelli vincolati sono fissati staticamente
 - e gli assegnamenti devono essere coerenti

S3: SPARK - C.Montangero - Copyright 2010

7

S3 2009/10 – SPARK – Moduli

OGGETTI E TIPI ASTRATTI

S3: SPARK - C.Montangero - Copyright 2010

8

Esempio: oggetto stack

- Definizione
 - di un modulo
 - allocato e inizializzato a tempo di caricamento (*elaborazione, in Ada*)
- No costruttori
- Ada ha
 - *generic package*
- Definizione in due parti:
 - specifica: interfaccia
 - body: realizzazione
 - con memoria

```
package Stack
is
function Empty return Boolean;
procedure Clear;
procedure Pop(X : out Integer);
procedure Push(X : in Integer);
end Stack;
```

S3: SPARK - C.Montangero - Copyright 2010

9

Esempio: tipo Stack

- Definizione
 - di un tipo
 - inizializzato a tempo di elaborazione
- Costruttori
 - possibili come funzioni
 - spesso costanti

```

    • body: realizzazione
      • senza memoria

package IntStack
is
  type IntStackT is private;
  EmptyIntStack : constant IntStackT;
  function Empty(S: IntStackT)
    return Boolean;
  procedure Clear(S: in out IntStackT);
  procedure Pop(S: in out Stack;
                X: out Integer);
  procedure Push(S: in out Stack;
                 X: in Integer);
end Stack;
    
```

S3: SPARK - C.Montangero - Copyright 2010

10

S3 2009/10 – SPARK – Moduli

MODULI COME OGGETTI

S3: SPARK - C.Montangero - Copyright 2010

11

Approccio

- ASM:
 - abstract state machine
- Sistema SPARK
 - ASM interagenti
 - tramite le componenti pubbliche dello stato
 - un solo flusso di controllo

S3: SPARK - C.Montangero - Copyright 2010

12

Moduli come oggetti: specifica

- Specifica delle operazioni
 - come sottoprogrammi che modificano lo stato
 - come denotarlo nelle derives?
 - annotazione per dichiarare le variabili che definiscono lo stato
 - in genere astratto -> una variabile
 - raffinamento nel body
- Raffinamento:
 - definizione delle strutture dati concreti
 - rappresentazione del tipo astratto
- Indipendenza spec/impl

S3: SPARK - C.Montangero - Copyright 2010

13

Esempio: Stack

- Il risultato di Empty dipende implicitamente da State
- Lo stato dopo clear non dipende da nulla
- Pop modifica lo stato, e il valore restituito dipende da State
- Lo stato dopo Push dipende dal valore inserito e dallo stato precedente
- own da Algol60

```

package Stack
--# own State;
is
  function Empty return Boolean;
  --# global State;
  procedure Clear;
  --# global out State;
  --# derives State from ;
  procedure Pop(X : out Integer);
  --# global in out State;
  --# derives State from *
  --# & X from State;
  procedure Push(X : in Integer);
  --# global in out State;
  --# derives State from *,X;
end Stack;
    
```

S3: SPARK - C.Montangero - Copyright 2010

14

Moduli come oggetti: realizzazione

- Corpo delle operazioni
 - come sottoprogrammi che modificano lo stato concreto
- Raffinamento: *rappresentazione* del tipo
 - # own State is Pointer, Vector;
 le variabili a destra sono dichiarate nel body (owned):

```

Pointer : PointerT; Vector : IntStackT ;
    
```

- nel corpo delle operazioni
 - le derives parlano di Pointer e/o Vector dove, nella specifica, si parla di State

S3: SPARK - C.Montangero - Copyright 2010

15

Annotazioni raffinate

```

procedure Pop(X : out Integer)
--# global in out Pointer; in Vector;
-- era in out State
--# derives Pointer from *
--# & X from Pointer, Vector;
-- era State from * & X from State;
is begin
X := Vector(Pointer); Pointer := Pointer - 1;
end Pop;
    
```

- Sulle singole parti si può restringere il flusso informativo
- Ma tra tutte si deve render conto del flusso astratto:
`global in Pointer, Vector; -- errato!`
- Le derives raffinate rendono conto dei nuovi flussi

S3: SPARK - C.Montangero - Copyright 2010

16

Inizializzazioni

- Lo stato di un package può essere inizializzato
 - al caricamento (*elaborazione*)
 - nelle singole dichiarazioni
 - nella clausola begin (alla fine del package)
 - più espressivo
 - dinamicamente (in esecuzione)
 - non accettato per stato usato da un main
- Esempi
 - in GPS (IntStack)

S3: SPARK - C.Montangero - Copyright 2010

17

S3 2009/10 – SPARK - Moduli

IMPORTAZIONI

S3: SPARK - C.Montangero - Copyright 2010

18

Accesso ai moduli esterni

- Clausole with e inherit
 - la seconda introdotta da SPARK
 - apre la visibilità alle annotazioni
 - accesso con ".": nome_package.nome_entità
- Prefisse alla parola chiave "package"
 - package annidati: no with, sì inherit
- "with" anche per body
 - inherit non serve
 - le nuove entità non possono essere nelle annotazioni
 - il main fa eccezione!

S3: SPARK - C.Montangero - Copyright 2010

19

Esempio: IntStack

```
with Stack;
--# inherit Stack;
--# main_program;
procedure MainInherit
--# global Stack.State;
--# derives Stack.State
--# from *;
is
  I : Integer;
begin
  -- Stack.Clear; -- inizializzazione dinamica
  -- non accettata

  Stack.push(25);
  Stack.pop(I);
  Stack.Push(I); -- uso di I...
end MainInherit;
```

S3: SPARK - C.Montangero - Copyright 2010

20

Esempio: interi su standard IO

```
with Spark_IO;
--# inherit Spark_IO;
package Standard_Spark_IO
...
procedure Std_Put_Integer (Item : in Integer );
--# global in out Spark_IO.Outputs ;
--# derives Spark_IO.Outputs
--# from *, Item;
```

- non cambio SPARK_IO
 - faccio una "façade" (facciata)

S3: SPARK - C.Montangero - Copyright 2010

21

Importazione: transitività?

```

with Stack,
    Standard_Spark_IO,
    Spark_IO;
--# inherit Stack,
--#     Standard_Spark_IO,
--#     Spark_IO;

--# main_program;
procedure Main
-- usa solo Standard_Spark_IO
-- ma questo usa Spark_IO
-- nelle sue annotazioni
...

```

- No transitività implicita
- Da esplicitare, se serve
- Standard... fornisce solo valori di default a sottoprogrammi di Spark_IO
- Le annotazioni riguardano lo stato di quest'ultimo

S3: SPARK - C.Montangero - Copyright 2010

22

Stack: c'è un problema, in SPARK?

- Stack overflow!
- Si può usare così solo se si dimostra / argomenta che
 - nessuna pop è fatta su stack vuoto
 - nessuna push è fatta su stack pieno
- Altrimenti firme con
 - parametro `esito_OK` : Boolean
 - esaminato al ritorno
 - lasciato come esercizio

S3: SPARK - C.Montangero - Copyright 2010

23

S3 2009/10 – SPARK - Moduli

TIPI DI DATI ASTRATTI

S3: SPARK - C.Montangero - Copyright 2010

24

Abstract Data Type (ADT)

- Idea: definire un tipo
 - definendo le proprietà delle operazioni
 - piuttosto che in base alla rappresentazione
- Esempio: push e pop definite implicitamente
 - emptyStack : Stack -- costante
 - pop(push(S,x)) = x
 - empty(emptyStack) = true
 - empty(Push(S,x)) = false
- Information hiding segue naturalmente

S3: SPARK - C.Montangero - Copyright 2010

25

ADT: specifica

- Nome del tipo dei valori astratti
 - clausola **private**
- Specifica delle operazioni
 - come sottoprogrammi/funzioni
 - oggetto astratto come parametro: p(a)
 - invece di a.p()
 - annotazioni derives naturali
 - no --# global per lo stato (portato dal parametro)
 - rappresentazione nella specifica (parte privata)
- Maggiore dipendenza spec-impl

S3: SPARK - C.Montangero - Copyright 2010

26

Esempio: Stack (parte pubblica)

```

package Stack is
  type IntStackT is private;          -- esiste
  EmptyIntStack : constant IntStackT; -- pure
  function Empty(S: IntStackT) return Boolean;
  -- il valore dipende implicitamente da S
  procedure Clear(S: out IntStackT);
  --# derives S from ;
  procedure Pop(S: in out IntStackT;
               X : out Integer);
  --# derives S from *
  --# & X from S;
  procedure Push(S: in out IntStackT;
                X : in Integer);
  --# derives S from *,X;
private                                -- fine della parte pubblica

```

S3: SPARK - C.Montangero - Copyright 2010

27

Esempio: Stack (parte privata 1)

```
private
  StackSize : constant Integer := 100;

  subtype PointerT is Natural
    range Natural'First .. StackSize;

  type IntVectorT is array (Integer range <>)
    of Integer;
  subtype StackIndexRange is Positive
    range 1 .. StackSize;
  subtype IntStackSupportT is
    IntVectorT(StackIndexRange);
```

S3: SPARK - C.Montangero - Copyright 2010

28

Esempio: Stack (parte privata 2)

```
type IntStackT is record -- ecco il tipo
  Pointer : PointerT;
  Vector : IntStackSupportT;
end record;
-- e il valore della costante
EmptyIntStack : constant IntStackT :=
  IntStackT'(Pointer => 0,
    Vector =>
      IntStackSupportT'(StackIndexRange => 0));
end Stack;
```

- Il tipo record è essenziale per "impaccare" la rappresentazione
- Il compilatore vede la parte privata e può allocare la memoria per gli oggetti di tipo "astratto"
- Nella documentazione si può nascondere la parte privata

S3: SPARK - C.Montangero - Copyright 2010

29

ADT: Realizzazione

- Dal body si vede anche la parte privata
- Non serve raffinamento
 - si usa il tipo privato
 - si devono rispettare comunque le annotazioni

```
function Empty (S: IntStackT) return Boolean is
begin
  return S.Pointer = 0; -- selezione campo
end Empty;

procedure Clear (S: out IntStackT) is
begin
  S := EmptyIntStack;
end Clear;
-- codice completo in IntStackADT
```

S3: SPARK - C.Montangero - Copyright 2010

30

ADT: uso

```
Inp,Outp : Integer;  
Outcome : Boolean;  
MyStack : Stack.IntStackT := Stack.EmptyIntStack;
```

```
begin  
  Standard_Spark_IO.Std_Get_Integer(Inp, Outcome);  
  if not Outcome then  
    Standard_Spark_IO.Std_Put_Integer(-1); --errore  
    return;  
  end if;  
  Stack.Push(MyStack, Inp);  
  Stack.Pop(MyStack, Outp);  
  Standard_Spark_IO.Std_Put_Integer(Outp);  
  Stack.Pop(MyStack, Outp);
```

- Come si legittima l'uso di Stack?
– con **with** e **inherit**

S3: SPARK - C.Montangero - Copyright 2010

31
