Sviluppo di Software Sicuro - S^3
SPARK – Comandi

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

Sommario

- Comandi (statements) elementari
- Strutture di controllo
  - Condizionali
  - Cicli
- Chiamate di sottoprogrammi
- Bonus: Input/output

S^3 2009/10 – SPARK – Controllo elementare

COMANDI ELEMENTARI

Nullità e sequenze

```
sequence_of_statements ::= statement {statement}
statement ::= simple_statement | compound_statement
simple_stamentent::= null_statement
        | assignement_statement | …

null_statement ::=  null ;
```

- Il comando nullo è a volte utile
  - per esplicitare l'assenza di operazioni
- Il **;** è un terminatore...

Assegnamento

```
assignement_statement ::= variable_name := expression ;
```

- Se il tipo del valore è incompatibile con quello della variabile, il programma termina:
  - nulla deve andar male
  - se c'è un'eccezione, tanto vale terminare.
- Attenzione con i tipi intervallo (range):
  - vedremo poi i controlli statici in merito

S^3 2009/10 – SPARK – Controllo elementare

STRUTTURE DI CONTROLLO

Panoramica

```
statement ::= simple_statement | compound_statement
compound_statement ::= if_statement
                     | case_statement
                     | loop_statement
```

- Compongono "sequence_of_statements"
- Ciascuno col suo "**;**" finale...
- Semantica attesa
  - "case" è da intendersi come switch,
  - fa l'analisi per casi

if_statement

```
::= if condition then
      sequence_of_statements
    { elseif condition then
      sequence_of_statements }
    [ else
      sequence_of_statements ]
    end if;
```

- Le condizioni sono espressioni booleane
- Valutate nell'ordine, ma
  - Le funzioni SPARK sono senza effetti collaterali
  - l'ordine è irrilevante, il compilatore può ottimizzare

Esempio: estremi di un campione

```
if X > Max then
      Max := X;
elseif X < Min then
      Min := X;
else
      null;
end if;
```

- Le parentesi intorno alla condizione ci son già...
  - diversamente da Java

case_statement

```
    ::= case expression is
          when discrete_choice_list => sequence_of_statements
        { when discrete_choice_list => sequence_of_statements }
        [ when others => sequence_of_statements ]
        end case;
discrete_choice_list ::= discrete_choice { | discrete_choice }
discrete_choice ::=
        static_simple_expression | discrete_range
discrete_range ::=
        static_simple_expression .. static_simple_expression
```

- I due punti per "intervallo" (range)
  - ci torniamo coi sottotipi

Esempio: la settimana

```
case Today is
  when Lun .. Gio => Lavora;
  when Ven => Lavora; Disco;
  when Sab | Dom => null;
end case;
```

- Deterministico
  - ogni caso scelto una sola volta
- Se le espressioni non sono statiche
  - ci vuole **others**, altrimenti SPARK protesta

loop_statement

```
                ::= [ loop_identifier : ]
                    [ iteration_scheme ] loop
                        sequence_of_statements
                    end loop [loop_identifier] ;

iteration_scheme :: =
        while condition | for loop_parameter_specification
exit_statement ::= exit [loop_identifier] [when condition]
```

- Exit fa uscire sempre e solo dal loop corrente
  - Il nome nell'exit è ridondante, per documentazione
  - exit da while e for deprecabile a S3
- for poi, legato tipi intervallo
- Per il resto, semantica attesa

Esempi

```
Ricerca:
loop
  Get(CarattereCorrente);
  exit Ricerca when CarattereCorrente = '*';
end loop Ricerca;
```

```
Get(CarattereCorrente);
while CarattereCorrente /= '*'
loop
  Get(CarattereCorrente);
end loop;
```


S³ 2009/10 – SPARK – Controllo elementare

SOTTOPROGRAMMI

Panoramica

- Due forme, due parole chiave
  - **procedure**
  - **function**
- No ricorsione (= no memoria dinamica)
- Approccio SPARK
  - assimilazione parametri – globali
    - stato interno del modulo (= variabile statica della classe)
    - stato visibile di un altro module (= variabile statica pubbliche di altre classi)
  - specifica dei parametri estesa
    - per dichiarare le globali
    - per analisi di flusso

Accesso ai parametri

- Comune alle due forme
- All'interno del sottoprogramma:
  - **in** : costante inizializzata dalla chiamata
  - **out** : variabile non inizializzata
  - **in out** : variabile inizializzata
    - in entrambi i casi out :
    - valore assegnato all'argomento
      - come risultato dell'esecuzione del sottoprogramma

Trasmissione degli argomenti

- Due forme
  - per copia (by-copy / value-result)
    - tipi elementari
  - per riferimento (by reference)
    - tipi che non ci interessano (Ada, non SPARK)
  - sia l'uno sia l'altro (a scelta dell'implementazione)
    - tipi compositi
- Possibile nondeterminismo
  - per aliasing / terminazioni anomale
  - Ada: programma erroneo
  - SPARK: no aliasing / no eccezioni
    - ergo deterministico

Procedure

- Scopo: aggiornare lo stato del sistema
  - parametri out
  - variabili globali
- Approccio SPARK
  - compito principale: modificare le globali
    - (parte dello) stato di un modulo
    - come i metodi in una classe
  - restituire informazioni ausiliarie
    - parametri out: informazioni sull'esito, ...

Funzioni

- Scopo: restituire un valore (anche composito)
  - no effetti collaterali (no modifica dello stato globale)
  - no parametri out
- Esempio: generare numeri casuali
  - spesso funzione con effetti collaterali
    - Integer random()
  - SPARK:
    - procedura con parametro out per il valore generato
      - globale in out per il seme
    - funzione con aggiornamento indipendente
      - cattivo progetto

Esempio: numeri casuali

```
procedure Random (X : out Float);
--# global in out State;
--# derives  X, State
--#      from  State;
```

- Random accede a
  - X, in modo out (indefinito all'inizio)
  - State, in modo in out (definito all'inizio)
- Calcola i valori finali (out) di
  - X, in funzione di State
  - State, in funzione di State

Contratti

- La specifica di Random è un *contratto*
- Più esplicito della sola signature
- Non solo tipi dei risultati
  - relazione **derives** tra ingressi e uscite

  => nel corpo ci devono essere assegnamenti sufficienti a garantire tale dipendenza
  - Examiner controlla staticamente
- Descrizione completa della relazione
  - pre-/post-condizioni dalla specifica formale
  - Examiner aiuta a provare la correttezza

Sintassi procedure

```
procedure_body ::=
        procedure  defining_identifier   [ parameter_profile ]
        procedure_annotation
        is
           declarative_part
        begin
           sequence_of_statements
        end designator ;
```

- SPARK vuole il nome ripetuto alla fine
  - **procedure** Random ... **end** Random ;

Parametri delle procedure

```
parameter_profile ::=
        ( parameter_specification { ; parameter_specification } )
parameter_specification ::=
        defining_identifier_list : mode  subtype_mark
mode ::= [ in ] | in out | out
```

- Simile a UML
  - sia sintatticamente sia semanticamente

Annotazioni delle procedure

```
procedure_annotation ::=
        [ global_definition ]
        [ dependency_relation ]

Le definizioni globali sono del tipo

--# global  in      V1;
--#         in      V2;
--#         in out  V3;
--#         in out  V4;
--#            out  V5

Le Vi devono essere visibili...
```

Annotazioni delle procedure (2)

procedure_annotation ::= … [dependency_relation]

Relazioni che esprimono la possibile dipendenza:

```
--# derives   V11, V12
--#    from    V21, V22
--# &          V3    from    V4;
```

Es: il valore esportato di V11 *può* dipendere da V21, V22,
ma da nient'altro (può essere costante)
Le Vi devono essere visibili...
Abbreviazioni e casi particolari

S3: SPARK - C.Montangero - Copyright 2010 25

Esempi estremi

```
procedure Nothing
--# derives   ;   -- non fa nulla (a volte utile come segnaposto)

procedure Wait(T : Time)
--# derives  null  from  T;     -- la dipendenza da T non si
                                 --manifesta in una modifica dello stato
procedure Clear(S : out Stack)
--# derives  S from ;    -- il valore prodotto non dipende dallo
                          -- stato visibile
```

- Giustificazione esplicita del (*mancato*) effetto della procedura.

S3: SPARK - C.Montangero - Copyright 2010 26

Esempi normali

```
procedure Main
--# global in out  State,  Outputs;
--# derives State      from State              &
--#           Outputs from State, Outputs ;

derives abbreviabile in
--# derives State, Outputs   from * , State
```

- "*" sostituito da ogni variabile nella lista di sinistra
- Semantica: dipende da sé stessa
- Assorbita se già presente a destra

S3: SPARK - C.Montangero - Copyright 2010 27

Esempi particolari

- AC (assegn cond)
  - modifica X (in funzione di I e sé stesso)
  - solo se I ha certe caratteristiche
  - segnala la modifica in M

```
procedure AC(X : in out Float; I: in Integer; M : out Boolean )
--# derives X from *, I & M from I;
```

  - X può passare indenne attraverso la procedura
  - senza assegnamenti

Esempi particolari (2)

- AC (assegn cond) *definisce* X (in funzione di X)
  - solo se I soddisfa la condizione

```
procedure AC(X : out Float; I: in Integer; M : out Boolean )
--# derives X , M from I;
```

- L'argomento per X può essere indefinito
  - può restare indefinito (Examiner dà solo warning)
  - dipende dalla politica di assurance adottata
- Dal punto di vista della sicurezza
  - Meglio non lasciare indefiniti, e chiedere l'inizializzazione al chiamante (in out)

Funzioni

- In SPARK sono pure
  - no effetti collaterali
  - un solo risultato: no parametri out
- Comando esplicito per restituzione risultato
  - ultimo del corpo della funzione

```
return_statement ::= return expression ;
```

- Tipo dell'espressione
  - compatibile con la dichiarazione della funzione

Sintassi funzioni

```
function_body ::=
        function defining_identifier parameter_and_result_profile
        [ global_definition ]
        is
           declarative_part
        begin
           sequence_of_statements
        end designator ;
```

- non ci sono dipendenze
- le globali son solo **in**, che si può sottintendere

Parametri delle funzioni

```
parameter_and_result_profile ::=
        [ formal_part ] return subtype_mark
formal_part ::=
        ( parameter_specification { ; parameter_specification } )
parameter_specification ::=
        defining_identifier _list : subtype_mark
```


Esempio

```
function F1(I: in Integer) return Integer
--# global G;
is begin
   return I;
end F1;
```

Cosa dice Examiner?

```
Flow Error      : 50: The function value is not derived from the
   imported value(s) of G.
```

Numeri casuali

```
function Random return Integer
--# global Seed;
is begin
  return ...Seed...;
end Random;
```

- passa il controllo di Examiner?

- L'aggiornamento su Seed fatto altrove
- Sintassi: no parentesi per no parametri (anche nelle chiamate)

S³ 2009/10 – SPARK – Controllo elementare

PROSSIMO ARGOMENTO:
SPARK – MODULI