TRADUZIONE DI

STATE AND TRANSITION DIAGRAM UML

E

CORRETTEZZA DEI PROGRAMMI

IN TLA+

a cura di:

Eleonora Antonelli
Simone Maletta
Stefano Novara

INDICE

1. GRAMMATICA DI TLA+

OPERATORI FONDAMENTALI

Tla+ è un linguaggio formale di specifica che descrive i sistemi in cui lo stato cambia in passi discreti ed è basato su TLA.

La sintassi è costruita utilizzando gli operatori della logica proposizionale, il linguaggio degli insiemi, gli operatori modali e quantificatori con dichiarazione di range.

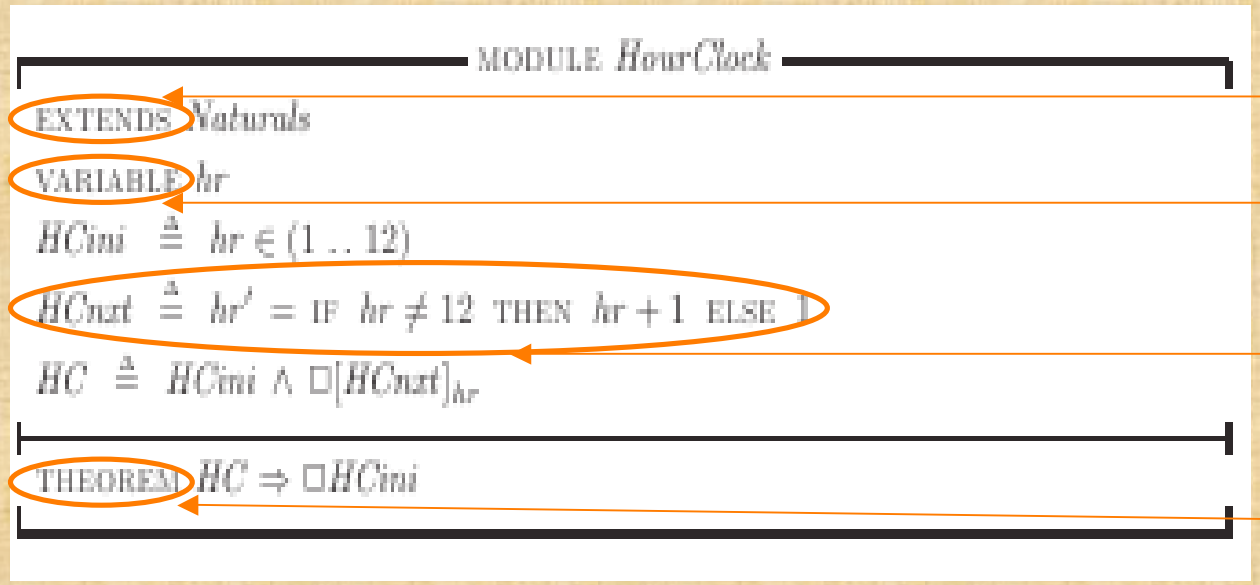
¬ NOT
V OR
Λ AND
→IMPLICAZIONE LOGICA
↔ EQUIVALENZA LOGICA
□ ALWAYS Se f è una formula logica, la formula temporale □ F asserisce che la formula F è sempre vera per ogni stato di ogni comportamento.
◊ EVENTUALLY Se f è una formula logica, la formula temporale ◊F asserisce che la formula F eventualmente è vera.
∀ QUANTIFICATORE UNIVERSALE
∃ QUANTIFICATORE ESISTENZIALE

- **ExistsIn (S, x: x + z > y)**
  Ci indica che: ∃ x ∈ S : x + z > y

GRAMMATICA DEI MODULI

.

TLA+ differisce da TLA semplicemente per la presenza dei moduli utilizzati per descrivere le specifiche.



Descrive i moduli che vogliamo utilizzare, in questo caso abbiamo importato il modulo dei Naturali.

Definisce le variabili che andremo ad utilizzare nel modulo.

Azioni: descrivono in che modo evolve il sistema. Il predicato iniziale: identifica tutti i possibili valori iniziali di hr

Indica una formula che deve essere vera in questo contesto.

Formalmente il modulo non identifica quale definizione è la specifica e quali sono le ausiliarie. Questo viene stabilito in un file a parte .cfg

ALTRI OPERATORI

- **Choose x ∈ S: p**

  Restituisce un valore v di S tale che p, sostituendo v ad x, è vera.

- **If p Then e1 Else e2**

  dove l'espressione è uguale a e1 se p è vero, mentre se p è falso è uguale a e2.

- **Case**

  E' utlizzato per semplificare l'espressione precedente sostituendo If/Then/Else con un unico operatore.

$$\text{CASE } p_1 \to e_1 \,\square\, \ldots \,\square\, p_n \to e_n$$

$$\text{CASE } p_1 \to e_1 \,\square\, \ldots \,\square\, p_n \to e_n \,\square\, \text{OTHER} \to e$$

$$\text{CASE } n \geq 0 \to e_1 \,\square\, n \leq 0 \to e_2$$

È uguale a e1 se n>0 è vero, è uguale a e2 se n<0 è vero e uguale a uno tra e1 o e2 se n=0 è vero.

GRAMMATICA DEI MODULI 2

- Inoltre in TLA+ possiamo utilizzare il concetto di **RECORD** dei linguaggi di programmazione.

Un insieme di record può essere descritto esplicitamente così:

chan ∈ [val: Data, rdy:{0,1}, ack:{0,1}]

L'ordine dei campi è irrilevante.

Quindi un possibile elemento **chan** potrebbe essere:

[val→d, rdy→1, ack→0]

Ed i campi possono essere estratti mediante l'istruzione:

chan.val, chan.rdy, chan.ack

- Infine in TLA+ possiamo usare anche il concetto di **TUPLA** che viene denotato con:

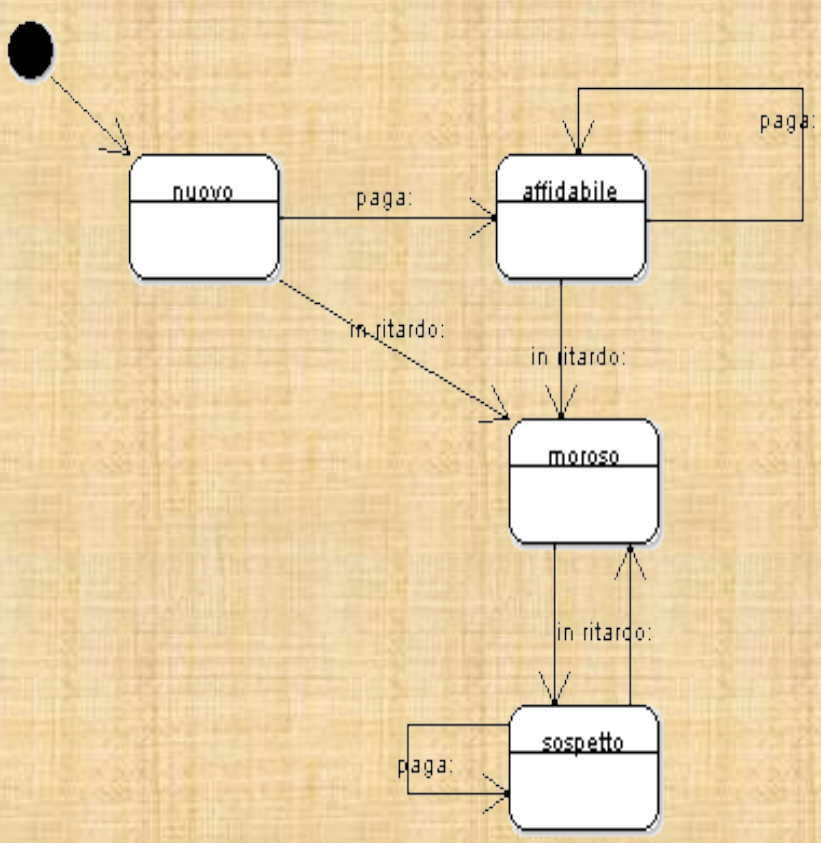
$$\begin{array}{lcl} \langle 1, 2, 3 \rangle & \in & Nat \times Nat \times Nat \\ \langle \langle 1, 2 \rangle, 3 \rangle & \in & (Nat \times Nat) \times Nat \\ \langle 1, \langle 2, 3 \rangle \rangle & \in & Nat \times (Nat \times Nat) \end{array}$$

Le 3 tuple non sono uguali ed x non è un operatore associativo.

- TLA+ definisce una **STRINGA** come una tupla di caratteri.

2. STATE AND TRANSITION DIAGRAM

DIAGRAMMA SENZA AZIONI



Traduzione degli stati

VARIABLES *stato;*

(mapping degli stati:*
stato=0 →nuovo
stato=1 →affidabile
stato=2 →moroso
stato=3 →sospetto
**)*

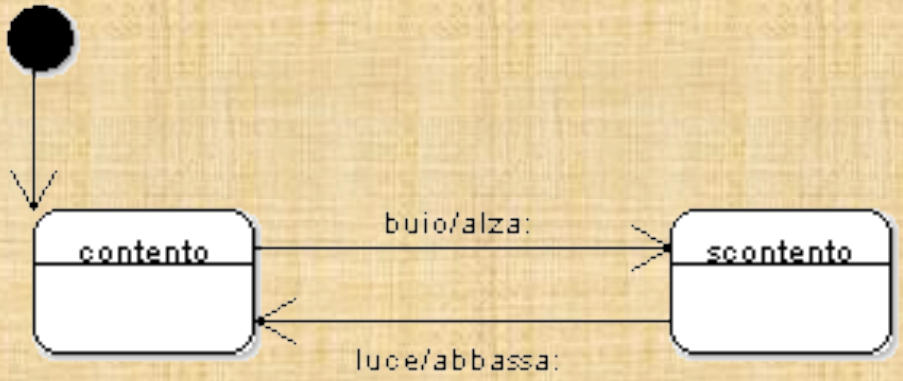
TYPE_INVARIANTS *stato∈(0,1,2,3)*

Traduzione delle azioni

paga≙ *stato'=CASE*
stato=0 →1 □
stato=1 →=1 □
stato=2 →3 □
stato=3 →3

ritardo ≙ *stato'=CASE*
stato=0 →2 □
stato=1 →2 □
stato=3 →2

SISTEMI CON AZIONI



Condizioni

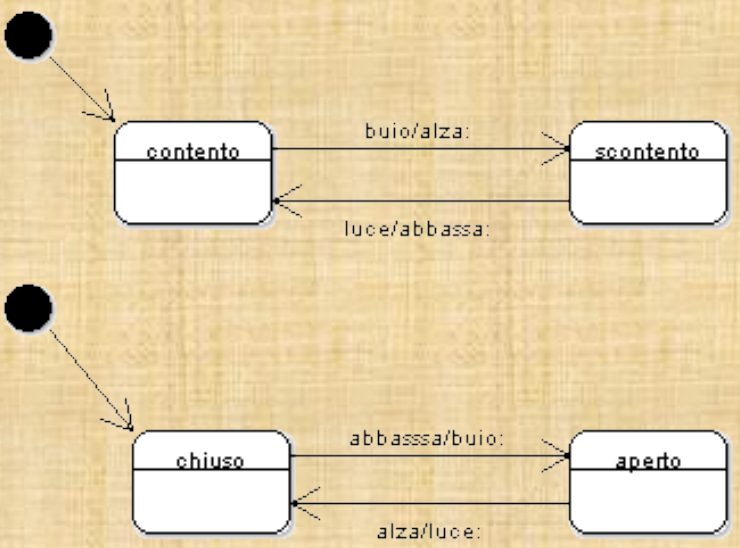
$$TYPE\_INVARIANTS \triangleq \quad \wedge stato \in \{0,1\} \quad \wedge cond \in \{0,1\}$$

Azioni

$$alza \triangleq \quad \wedge cond' = CASE\ cond=0 \rightarrow 1 \ \square\ TRUE \rightarrow cond$$

$$abbassa \triangleq \quad \wedge cond' = CASE\ cond=1 \rightarrow 0 \ \square\ TRUE \rightarrow cond$$

SISTEMI SINCRONI



abbassa→buio →alza →luce

abbassa → 00
buio → 01
alza → 10
luce → 11.

Interruttore

(* AZIONI *)

LOCAL luce ≙ ∧ (cond1'= CASE (stato=0 ∧ cond1=1 ∧ cond2=0) → 0
TRUE → cond1)
∧ (cond2'= CASE (stato=0 ∧ cond1=1 ∧ cond2=0) → 0
TRUE → cond2)

LOCAL buio ≙ ∧ (cond1'= CASE (stato=1 ∧ cond1=0 ∧ cond2=0) → 1
TRUE → cond1)
∧ (cond2'= CASE (stato=1 ∧ cond1=0 ∧ cond2=0) → 0
TRUE → cond2)

(* EVENTI *)

alza≙ (stato'=CASE (stato=0 ∧ cond1=1 ∧ cond2=0) → 1 TRUE → stato)
∧ luce

abbassa ≙ (stato'=CASE (stato=1 ∧ cond1=0 ∧ cond2=0) -> 0 TRUE → stato)
∧ buio

Next ≙ ∨ alza ∨ abbassa

Bambino

(* AZIONI *)

LOCAL alza ≙ ∧ (cond1'=CASE (stato=0 ∧ cond1=0 ∧ cond2=1) →1 ∈
TRUE → cond1)
∧ (cond2'=CASE (stato=0 ∧ cond1=0∧cond2=1) →0 ∈
TRUE → cond2)

LOCAL abbassa ≙ ∧(cond1'=cond1)
∧(cond2'=CASE (stato=1 ∧ cond1=1 ∧ cond2=1) →1 ∈
TRUE → cond2)

(* EVENTI*)

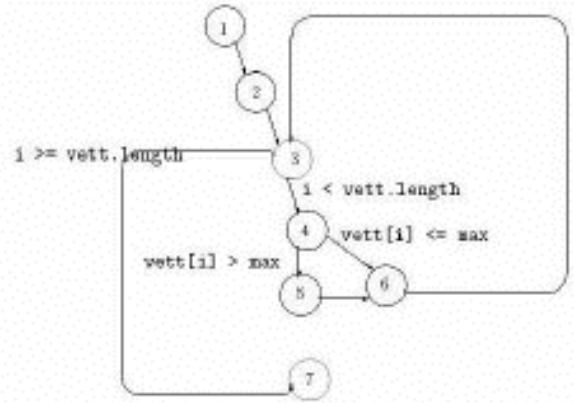
buio ≙ ∧(stato'= CASE (stato=0 ∧ cond1=0 ∧ cond2=1)→1 ∈TRUE → stato)
∧ alza

luce ≙ ∧(stato'= CASE (stato=1 ∧ cond1=1 ∧ cond2=1)→ 0 ∈TRUE → stato)
∧ abbassa

Next ≙ ∨ buio ∨ luce

PROGRAMMI

```
public static int Massimo(int [] vett)
{
     int result= MIN;                          /*1*/
     int i=0;                                  /*2*/
     while(i<vett.length)                      /*3*/
     {
          if(vett[i]>result)                   /*4*/
               result=vett[i];                 /*5*/
          i++;                                 /*6*/
     }
     return result;                            /*7*/
}
```



Massimo

EXTENDS Naturals, Sequences

VARIABLES i, stato, result

CONSTANT MIN,N

(* INIZIALIZZAZIONE *)

vett ≜ <<3,15,6>>

Minv≜ ∧ i ∈ Nat
∧ stato ∈ {1,2,3,4,5,6,7}
∧ N ∈ Nat
∧ N>0

MInit≜ ∧ result=MIN
∧ i=1
∧ stato=3

(* TRANSIZONI DEL GRAFO *)

LOCAL testWhile ≜ ∧ (stato'= CASE (stato=3 ∧ (i = <N)) → 4
(stato=3 ∧ (i > N)) → 7
TRUE → stato)
∧ result'=result
∧ i'=i

LOCAL testIf≜ ∧ (stato'= (CASE (stato=4 ∧ (vett[i]> result))→5
(stato=4 ∧ (vett[i] \leq result))→6
TRUE → stato))
∧ result'=result
∧ i'=i

LOCAL assegnazione≜ ∧ (stato'=(CASE stato=5 →6
TRUE→ stato))
∧ (result'= (CASE stato=5→ vett[i]
TRUE→result))
∧ i'=i

LOCAL incI≜ ∧ (stato'= CASE (stato=6)→3
TRUE → stato)
∧ (i'=CASE (stato=6)→ i+1
TRUE →i)
∧ result'=result

LOCAL return ≜ ∧ stato'= stato
∧ result'=result
∧ i'=i

Next ≜ testWhile ∨ testIf ∨ assegnazione ∨ incI ∨ return

Mspec≜ MInit ∧ $\Box$[Next]$_{<result,i,stato>}$ ∧ (i ≤ N)

---

3. TESTING AUTOMATICO

IL TOOL TLC

TLC è un model checker utilizzato per vagliare specifiche ridotte in linguaggio TLA+; questo riceve in input un file che riporta i moduli TLA+ per compiere model checking su di esse.
I file che questo tool prende in input sono scritti in un linguaggio ASCII molto simile a TLA+, differisce in realtà la sola rappresentazione figurativa dei simboli, ad esempio il simbolo di definizione viene scritto come == e non come $\triangleq$, oppure il simbolo di diverso è scritto come #.
Altro input di TLC è un file di configurazione, che non possiede alcuna collusione colla logica di Lamport, che in sostanza indica al TLC i nomi delle specifiche e delle proprietà da verificare.
Il modo più efficace di trovare errori in una specifica è quello di provare a verificare che questa soddisfi le proprietà desiderate.
Possiamo utilizzare TLC senza dover verificare alcuna proprietà, nel qual caso esso andrà a cercare solo due tipi di errore: errori di *Silliness, cioè errori di semantica, oppure errori di Deadlock, dei* quali il TLC cerca di dimostrarne l'assenza.
Le specifiche e le proprietà che TLC verifica sono mere formule temporali.
Nella nostra trattazione, abbiamo preso in esame gli esempi di casi di studio proposti precedentemente in traduzione utilizzando il tool per verificare la correttezza delle nostre specifiche e dimostrare alcune proprietà, relative alle stesse.