ALTARICA + Abstract Data types: The Persee model

MVTsi team

LaBRI

Outline

- The ALTARICA_PERSEE Model.
- Description.
  - The ALTARICA model
  - New features.
  - Abstract Data types.
- Deliverables.
- An Example.

The ALTARICA_PERSEE model

Definition *An* ALTARICA_PERSEE *Constraint Automaton:*

- *two sets of variables* S, F*,*
- *a set of events* E *with a special one* ϵ*.*
- *a set of transitions :*

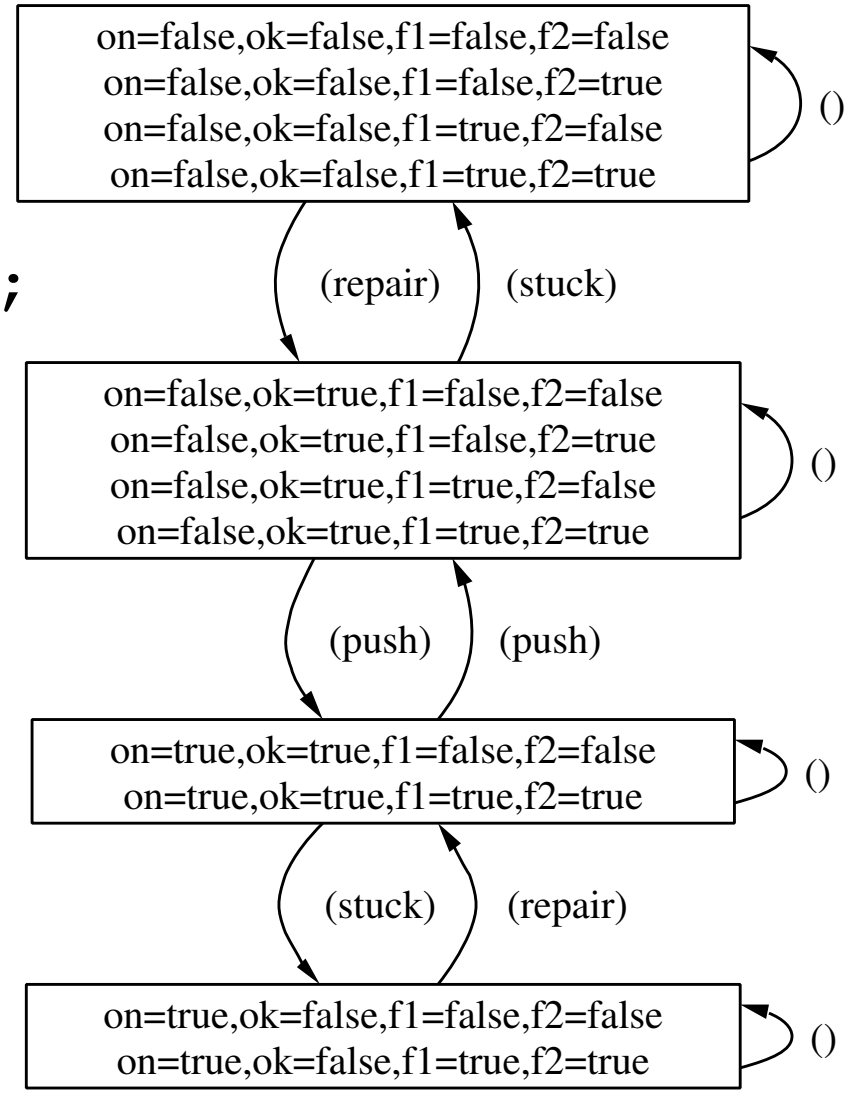
$$G(\vec{s}, \vec{f}) \xrightarrow{e} \vec{s} := \sigma(\vec{s}, \vec{f})$$
$$True \xrightarrow{\epsilon} \vec{s} := \vec{s}$$

  $G(\vec{s}, \vec{f})$ *is a guard (i.e. a boolean formula) and* $e \in E$*,*

- *an assertion* $A(\vec{s}, \vec{f})$*, acts both as pre and post conditions*
- *partial order* $\prec_E$ *for priorities.*

The ALTARICA model (1)

```
node Switch
  flow   f1, f2 : bool;
  state  on, ok : bool;
  event
    push, stuck, repair;
  trans
    ok  |- push   ->
           on := ~on;
    ok  |- stuck  ->
           ok := false;
    ~ok |- repair ->
           ok := true;
  assert on => (f1=f2);
  init   on := true;
edon
```



The ALTARICA model (2)

```
node Circuit
   sub     G : Generator;
           S : Switch;
           L : Lamplight
   assert  S.f1 = G.f1;
           L.f1 = S.f2;
           L.f2 = G.f2;
   sync    <G.failure, S.stuck>;
           <G.repair, S.repair>;
edon
```

Events are asynchronous by default, so we have to list synchronous events.

New features

- predefined domain: `integer`
- array domain: `token : bool[7][param]`
- structured domain:
  `Struct data : integer; parity : bool Tcurts`
- domain name:

```
message = Struct data : integer;
                parity : bool Tcurts
```

- parameters:
  `Param P1, P2 : domain; P3 : domain_name`
  `Param_set P1.data := 3, P1.parity := true`
- quantified expression:
  `<x,y,z : bool>(x = (y | z))` and
  `[a : integer](a div 3 = 0)`

Abstract Data types

- signature definition:

```
Sig FIFO
 queue   : FIFO * message -> FIFO;
 dequeue : FIFO -> FIFO;
 head    : FIFO -> message;
 empty   : FIFO -> bool;
Gis
```

- functional operators.
  side effects $\leftrightarrow$ parallel assignments.
- partial functions.
  domains are implicit $\rightarrow$ no rewriting rules.

The division example (1)

```
[mec] R1(x:[0,1], y:[0,1], z:[0,1]) :=
.       ~(x = y/z);
R1: ([ 0, 1 ], [ 0, 1 ], [ 0, 1 ]) -> bool
[mec] :display R1
(0, 1, 1)
(1, 0, 1)
(1, 1, 0)
(0, 1, 0)
(1, 0, 0)
(0, 0, 0)
```

The division example (2)

```
[mec] R2(x:[0,1], y:[0,1], z:[0,1]):=
.       ~(x = y/z & (z != 0));
R2: ([ 0, 1 ], [ 0, 1 ], [ 0, 1 ]) -> bool
[mec] :display R2
(0, 1, 1)
(1, 0, 1)
(1, 1, 0)
(0, 1, 0)
(1, 0, 0)
(0, 0, 0)
[mec] R3(x:[0,1], y:[0,1], z:[0,1]) :=
.       ~(x = y/z) & (z != 0);
R3: ([ 0, 1 ], [ 0, 1 ], [ 0, 1 ]) -> bool
[mec] :display R3
(0, 1, 1)
(1, 0, 1)
[mec]
```

Deliverables

- The syntactic tree: `http://altarica.labri.fr/Doc/Syntax/altarica-syntax.[ps|tex]`
- A parser for ALTARICA_PERSEE : `CVS altarica.labri.fr`

An example (1)

```
Const
 N = 4;
Domain
 message = Struct data : integer;
                  parity : bool Tcurts

Sig FIFO
 queue   : FIFO * message -> FIFO;
 dequeue : FIFO -> FIFO;
 head    : FIFO -> message;
 empty   : FIFO -> bool;
Gis
```

An example (2)

```
node Producer
 flow  m : message;
 event send
 trans true |- send ->;
edon

node Consumer
 state f : FIFO;
 flow  m : message;
 event receive
 trans (m.data mod 3)=0 |- receive ->
       f := queue(f,m);
edon
```

An example (3)

```
node Systeme
 state f : FIFO;
 sub P : Producer;
     C : Consumer[N];
 event in, out;
 trans true |- in -> f := queue(f,P.m);
       ~empty(f) |- out -> f := dequeue(f);
 sync
     <P.send, in>;
     <out, C[0].receive>;
     <out, C[1].receive>;
     <out, C[2].receive>;
     <out, C[3].receive>;
 assert
     empty(f) |
     (C[0].m = head(f) & C[1].m = head(f)&
      C[2].m = head(f) & C[3].m = head(f));
edon
```