N-way conformation

Danil Sokolov, Victor Khomenko, Alex Yakovlev

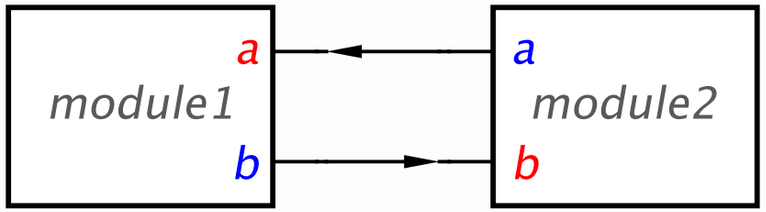
Newcastle University, UK

Conformation

- The circuit modules never break their environment by producing unexpected outputs
  - If at some state a module produces an output event $x+$, then at that state a transition labelled $x+$ must be enabled in the environment

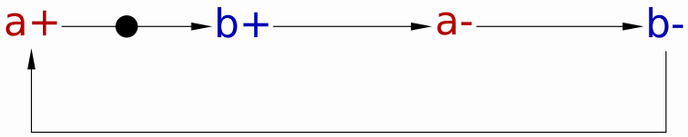
- Workcraft verification support
  - Conformation of Circuit model to its environment STG
    Verification $\rightarrow$ *Conformation [MPSat]*
  - Conformation of STG model to its environment STG
    Verification $\rightarrow$ *1-way conformation (1st STG without dummies) [MPSat]...*
  - N-way conformation of STGs
    Verification $\rightarrow$ *N-way conformation (without dummies) [MPSat]...*

Conformation: 2-module system

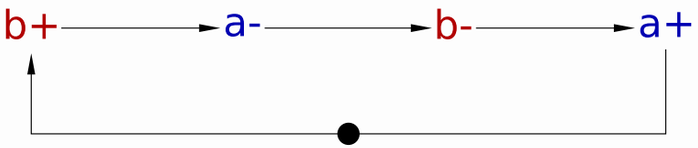


- Conformation check
  - `module1` conforms to `module2`
  - `module2` conforms to `module1`

- Conformant modules
  - STG for `module1`

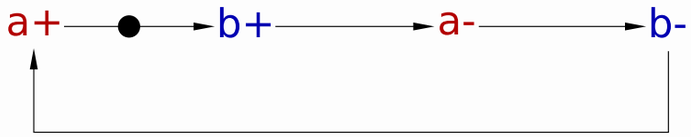


  - STG for `module2`

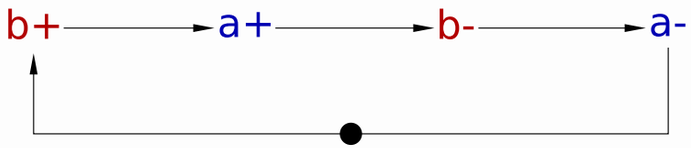


  - Result of conformation check:
    both modules conform to each other

- Non-conformant modules
  - STG for `module1`

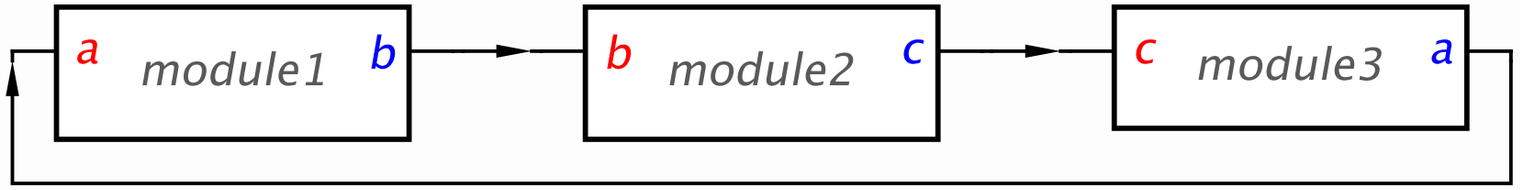


  - STG for `module2`



  - Result of conformation check
    `module2` breaks `module1` by unexpected `a+`

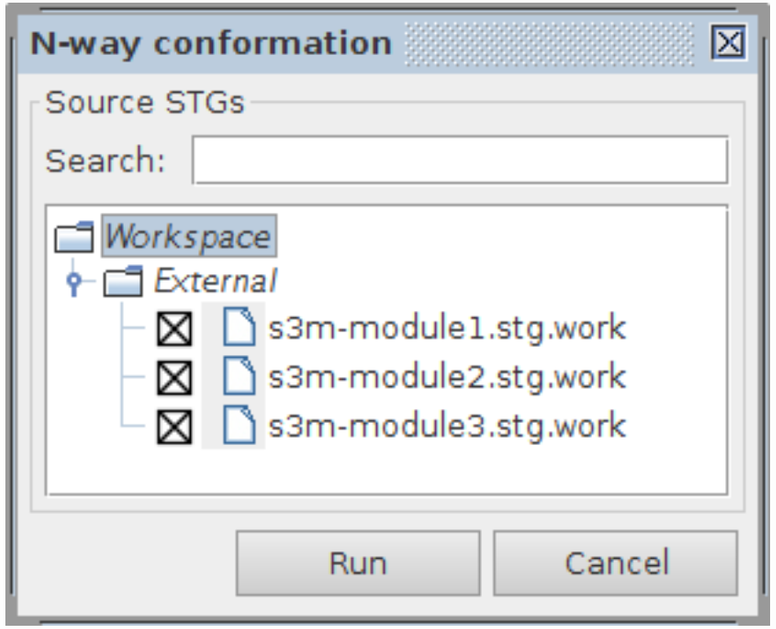
Conformation: 3-module system

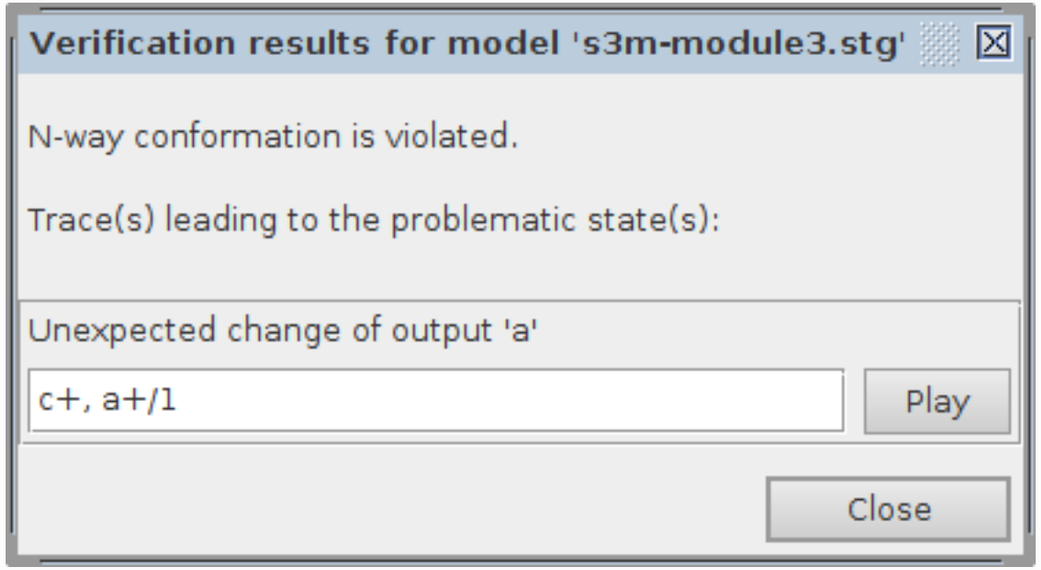


- Conformation check
  - `module1` conforms to composition of `module2` and `module3`
  - `module2` conforms to composition of `module3` and `module1`
  - `module3` conforms to composition of `module1` and `module2`
- Too much boilerplate – need for design automation!

N-way conformation

- Automatically checks conformation of each STG to the composition of the other STGs
- Uses a single parallel composition of all STGs
- Performs a single run of UNFOLDINGTOOLS toolchain
- Available for STG models via
  Verification → N-way conformation (without dummies) [MPSat]...





Reach property for N-way conformation

```
card DUMMY != 0 ? fail "Conformation can currently be checked only for device STGs without dummies" :
let
   SETS_OF_PLACE_NAMES = {
      {"<a+,b+>", "<a-,b->", "<b+,a->", "<b-,a+>", "#0"},
      {"<c+,a->", "<a-,c->", "<c-,a+>", "<a+,c+>", "#1"},
      {"<c+,b->", "<c-,b+>", "<b+,c+>", "<b-,c->", "#2"},
      {""}} \ {{""}},
   SETS_OF_OUTPUTS_NAMES = {{"b", "#0"}, {“a", "#1"}, {"c", "#2"}, {""}} \ {{""}},
   EXTENDED_PLACES = PP ".*@[0-9]+"
{
   exists PNAMES in SETS_OF_PLACE_NAMES {
      let
         TAG_SINGLETON = gather str in PNAMES s.t. str[0..0]="#" { str },
         OUTPUTS_SINGLETON=gather OUT_S in SETS_OF_OUTPUTS_NAMES s.t. card (TAG_SINGLETON * OUT_S) != 0 { OUT_S },
         PSTG = gather nm in PNAMES s.t. nm[0..0]!="#" { P nm },
         PSTG_EXT = PSTG + gather p in EXTENDED_PLACES s.t.
         let name_p=name p, pre_p=pre p, post_p=post p, s_pre_p=pre_p \ post_p, s_post_p=post_p \ pre_p {
            exists q in PSTG {
               let name_q=name q, pre_q=pre q, post_q=post q {
                  name_p[..len name_q] = name_q + "@" &
                  pre_q \ post_q=s_pre_p & post_q \ pre_q=s_post_p
               }
            }
         }
         { p },
         TSTG = tran sig (pre PSTG + post PSTG)
      {
         exists t in TSTG, OSTG in OUTPUTS_SINGLETON s.t. name sig t in OSTG {
            forall p in pre t s.t. p in PSTG_EXT { $p }
            &
            ~@ sig t
         }
      }
   }
}
```