Algorithm for calculating imprecise signatures

D. Krpelík

1 Notation

- number of components N
- number of types K
- number of components of type k in the system M_k
- state space $\Omega = \{0,1\}^N$
- Type State mapping : $L : \Omega \to \mathbb{N}^K$; maps the state to number of functioning components of each type
- Survivors $\mathbb{S} : \{0,1\} \times \mathbb{N}^K \to \mathbb{N}$, (number of unique states)

$$\mathbb{S}_i(l) := \#\{x \in \Omega : \varphi(x) = i \wedge L(x) = l\} \tag{1}$$

- The Survival signature

$$\Phi(l) := \frac{\mathbb{S}_1(l)}{\mathbb{S}_0(l) + \mathbb{S}_1(l)}, \tag{2}$$

2 the algorithm

2.1 Overall scheme

We will branch. At each step, the position in the branching scheme is tracked by two arrays:

- 'ones': a set which tracks which components are certainly functional,
- 'zeros': a set which tracks which components are certainly failed.

The accumulated 'survivors' (counts) will be held in structures 'sig0' and 'sig1', and converge to:

$$\text{sig0} \to \mathbb{S}_0, \quad \text{sig1} \to \mathbb{S}_1.$$

For each branch, we:

1. Find a minimal path, $\mathbb{MP}$, in the subgraph G, the original RBD, without the nodes in 'zeros'.
2. Account for all the states in Ω which will result in certain functionality of the system, those for which 'ones' $\cup\, \mathbb{MP}$ are functional. I.e. increase the counters held in 'sig1' (sec. 2.2).
3. Check, for each $p \in \mathbb{MP} -$ 'ones', whether $G -$ 'zeros' $- p$ can be functional (if the subgraph is connected). If not, increase the counter of 'sig0' (sec. 2.2).
4. Update the 'ones' with all the $p \in \mathbb{MP}$, which result into a cut set (so that we won't waste resources again in later branches).
5. For each $p \in \mathbb{MP}$, which did not result into a cut set and are not in 'ones', create a new branch, s.t. for arbitrary fixed indexing(ordering) i of (all) $p \in \mathbb{MP}$:
    - 'ones'$_i$ = 'ones' $\cup \bigcup_{j<i} p_j$,
    - 'zeros'$_i$ = 'zeros' $\cup\, p_i$.
6. Iterate for newly created branches.

Proposition 1 *Found minimal path (step 1) is an unique element of the RBD's minimal path set.*

Proposition 2 *If $G - zeros - p$ is not connected (step 3), then $zeros \cup p$ is a cut set (but not necessarily minimal). Additional computation would be needed if we would need the* **minimal** *cut set.*

Proposition 3 *The decomposition according to the found minimal path (step 5) is total and disjoint. It will decompose the space like '0xxxxxx', '10xxxxx', '110xxxxx',..., where x denotes 'arbitrary'.*

Proposition 4 *Similar branching could be done (not tested) by finding the minimal cut instead of the minimal path. This would result into a direct construction of the minimal cut set.*

2.2 Increase counters

(This is the same routine that Sean Reed uses in his BDD paper.)

- We have the 'ones' and 'zeros' sets, which represent components for which the state is already certain.
- Let $M'(k) = M(k) -$ amount of components of type k in 'ones' and 'zeros' vectors.
  I.e. how many components of type k can still obtain arbitrary state at the current position in the branching.
- For each vector $0 \leq x \leq M'$:
  1. $y := L(\text{'ones'}) + x$ ... a bit cryptic, but this simply computes the l vector s.t. all components in 'ones' are functional + increased by x_k in each k.
  2. 'sig0/1'$(y) += \prod_k^K \binom{M'(k)}{x(k)}$. Which adds how many states there are for which 'ones' are functional, so are another x, and 'zeros' are not.

3 Output

Whereever we stop in the branching process,

$$\frac{\text{`}sig1'(l)}{\prod_k \binom{M(k)}{l(k)}} \leq \Phi(l) \leq \frac{\prod_k \binom{M(k)}{l(k)} - \text{`}sig0'(l)}{\prod_k \binom{M(k)}{l(k)}},$$

with equalities, if we let the algorithm finish.