Possible Improvements For Modular Relative Time Petri Nets

(or: rooting for the underdog)

Contents

• Revision
• Helping or Hiding?
• Truncation
• Modular Truncation
  – Exhaustion, Prediction, Saturation, Prohibition
• Conclusion
Revision

• State space explosion is a problem
• Modular independence is a desirable design goal
• Timed systems are a desirable design goal
• Will it be even worse for both at once?

Helping or Hiding?

• Relative time vs Absolute time cost?
  – Relative time = ageing + firing
  – Absolute = firing
• Are we bailing more water just by having a bigger bucket?
  – ie. Why not just use absolute + congruence?
Helping or Hiding? (cont’d)

• “Lazy” Relative Time Petri Nets:
  – Same idea, but we are less stringent in ageing
• Now have two firing rules:
  – Normal = age(marking) – inputs + outputs
  – Lazy = marking – inputs + delay(outputs)
• Now we can decide the time we spend between exploring vs looking for cycles.

Truncation

• Truncation is based on only observable behaviours being interesting.
• If ageing a token will not change the net’s behaviour then don’t bother ageing it.
  – We call this the omega limit for a place.
• This prevents the “long tail” effect in timed systems (ie. an infinite divergence)
Modular Truncation

• Problem: global time vs relative time?
• Solution: keep the local systems in relative time still, apply the global time only at the points it is necessary.
• Problem: we still risk the “long tail” problem for any non-trivial system.
• Solution: modular time truncation!

Modular Truncation (cont’d)

• Same rule: if ageing a modules time stamp will not change the system’s behaviour…
• But when is this true?
  – When the local activity is exhausted
  – When the local activity is predictable
  – When the local activity is saturated
  – When the local activity is prohibitive
Exhaustion

• When the global time has advanced so far that the module must be deadlocked by now
• Can it really be that easy?
• Not quite, but almost!
• We need to remember that the rule is about possible behaviours, not just the next one.

Exhaustion (cont’d)

• A locally deadlocked process may have other timers waiting to expire as well!
• So, our exhaustion maximum will be:
  – $\max(\max(\text{omega}), \max(\text{path}_{\text{src}} + \text{omega}_{\text{dest}}))$
Prediction

• What if there is a local cycle that can be performed while the module waits?

• Sounds like a relative time congruence:
  – We reduce the age of the time stamp to the previous age with an identical local marking

• Similar to exhaustion, but we only take the *shortest* path for each identical destination.

Saturation

• What about non-deterministic and/or multiple local cycles?

• Be fair – this is a messy *model*, but we will still do our best to help it out. 😊

• We need to determine the reachable states:
  – $smear(src,t) = \{dest \mid \text{clock([src,…,dest])} = t\}$

• If the smear set is stable, we stop ageing.
Prohibition

- Some modules may stop participating forever (either by design or by accident).
  - i.e. all paths at least that long are strictly local
- The previous cases will still cover this, but prediction (for example) still wastes time.
- A prohibited module should have its timer *switched off*, once it is completely local.

Modular Truncation (cont’d)

- But this seems costly!
  - Are we making things *worse*?  

- Except that these properties…
  - are *local* and *static*!

- And a finite number is better than infinity!
Conclusion

• What is the relevance of RTPNs?
  – Can we do something *faster* than anyone else?
  – Can we do something *new* to anyone else?

• Can we “spend time to make time”?
  – Extra firing time vs wasted exploration time
  – More calculations up front vs fewer per step