UNIVERSITY of York

Synthesizing Real-Time Schedulability Tests using Evolutionary Algorithms: A Proof of Concept

Piotr Dziurzanski, Robert I. Davis, Leandro Soares Indrusiak

Real-Time Systems Group
Department of Computer Science
University of York
United Kingdom

RTS York

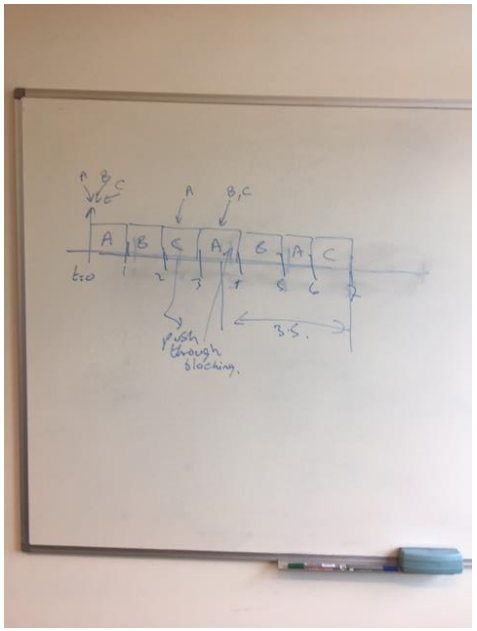
Introduction

- Real-Time Systems
  - Characterised by the need for both functional and timing correctness
- Verifying timing correctness: Typically two stage process
  - **Timing analysis** – characterises the amount of time each task can take to execute, or message can take to be transmitted
  - **Schedulability analysis** - aims to characterise the worst-case end-to-end response time of functionality involving one or more tasks or messages, taking into account the way in which they are scheduled
- How are schedulability tests derived?

How are schedulability tests derived?

Lots of thinking?

Trying out different equations using pen and paper?



Checking simple schedules on a whiteboard

- How do you do this?

Motivation for this work

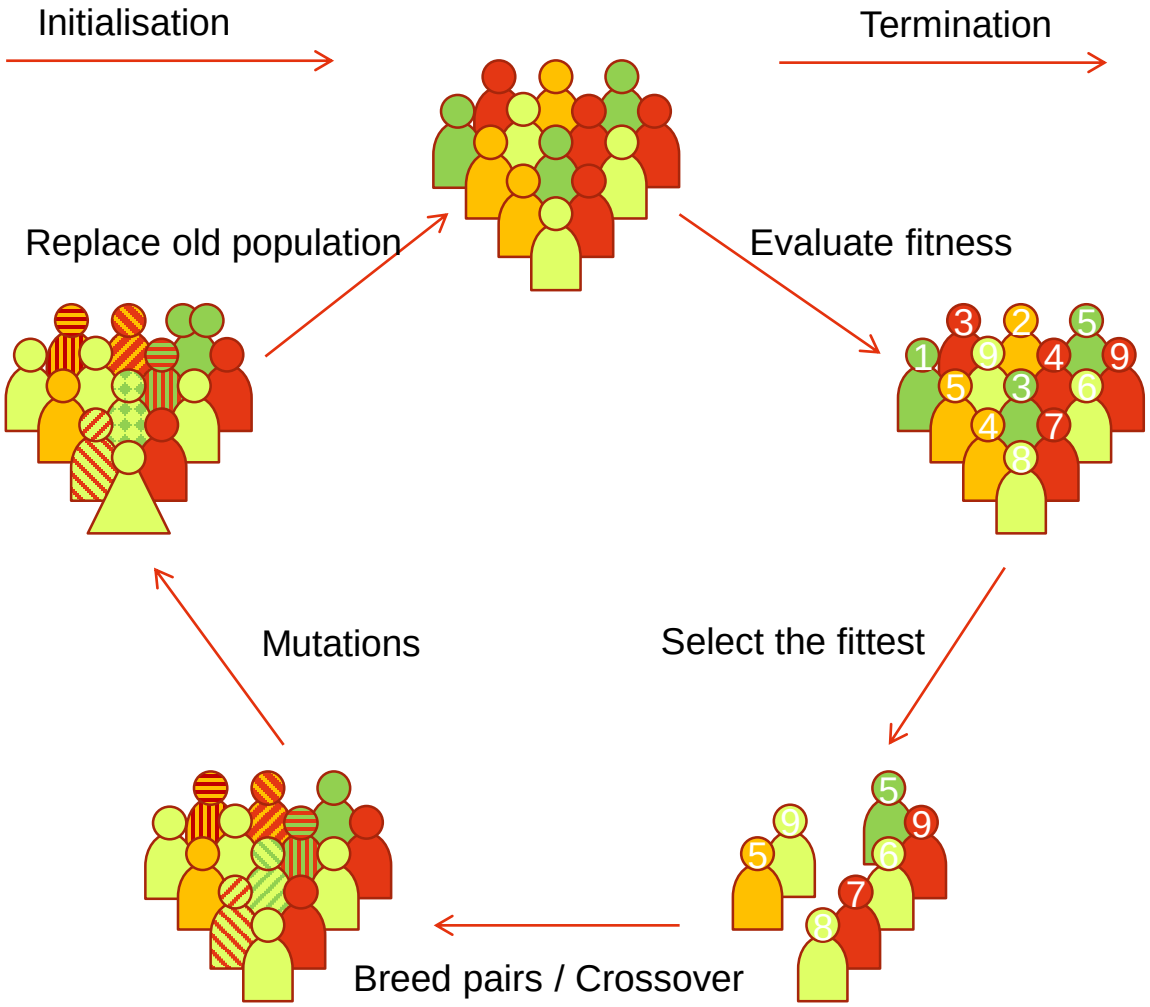
- Researchers
  - Have produced lots of great work on schedulability analysis
  - Also get things wrong
- Schedulability analysis literature many flawed proofs
  - Analysis for CAN from 1994/5 - flaws discovered and fixed in 2007
  - Analysis for self-suspending tasks from 2010 – flaws discovered in 2015 much subsequent theory also shown to be flawed in 2018
  - Analysis of wormhole routing on Network-on-Chip from 2008 – flaws discovered in 2016, revised analysis aiming to fix the problem also flawed!
- Work on formal proof can help

But we need to come up with analysis in the first place

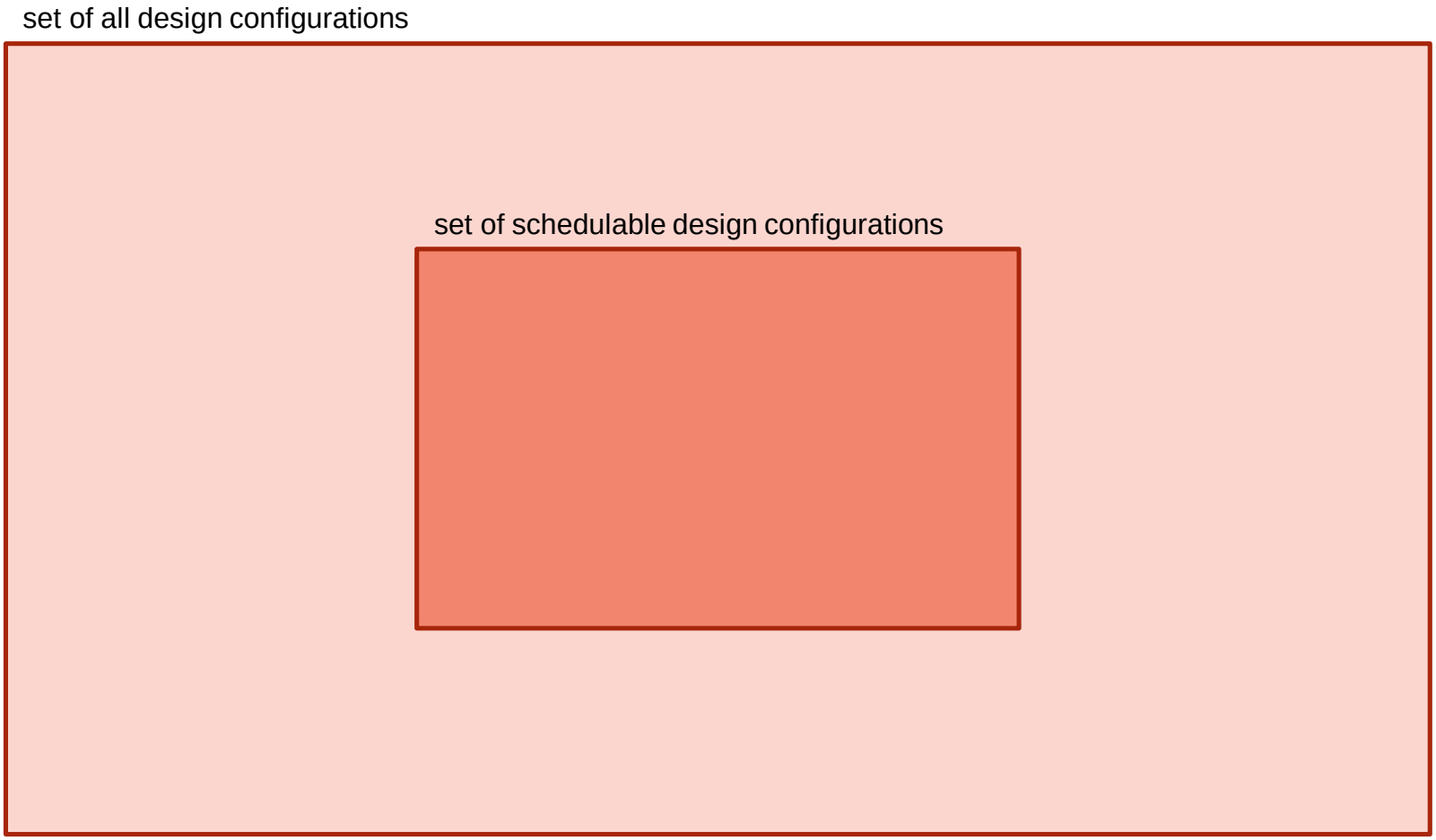
The Idea

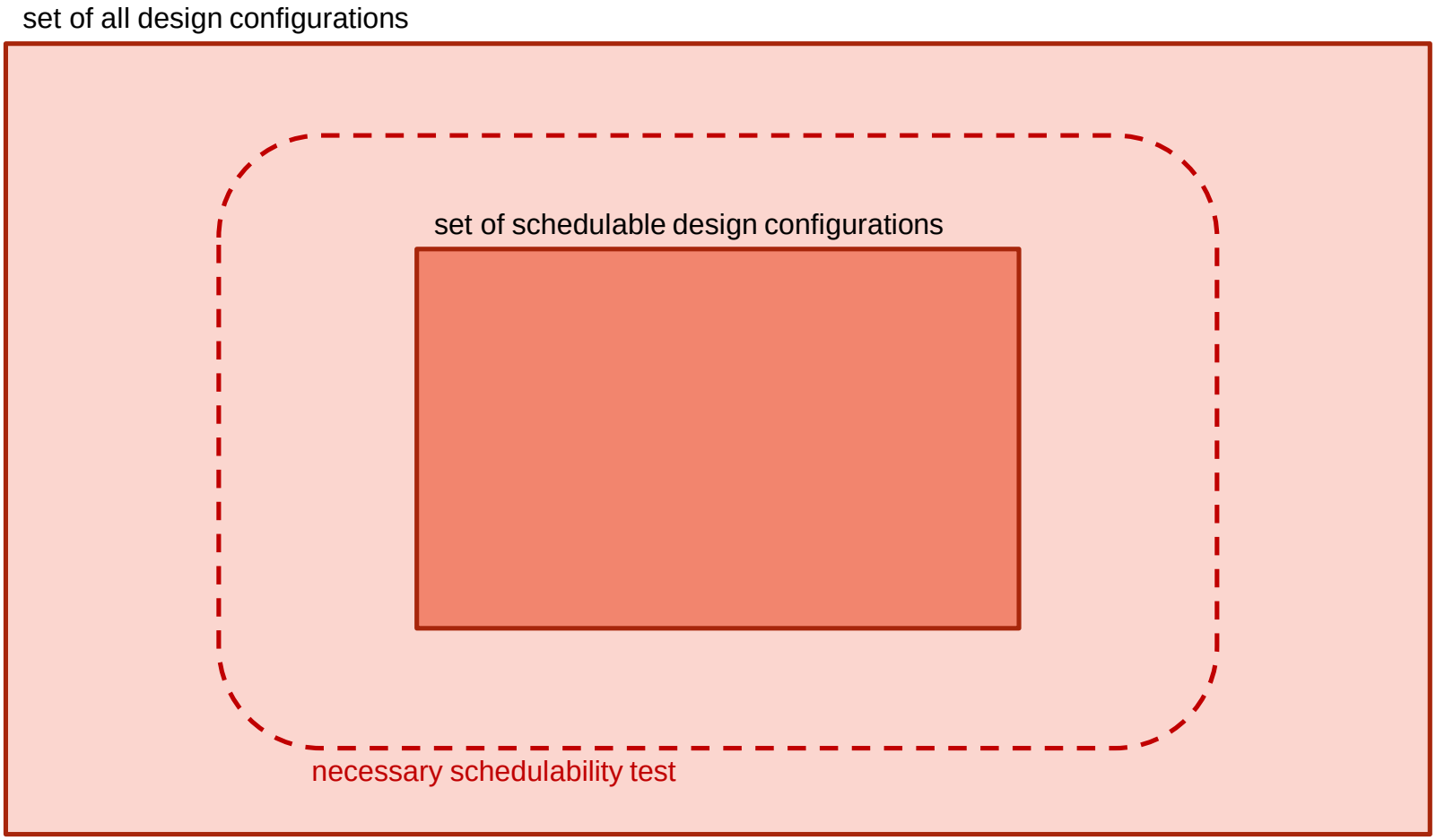
- Formulation Assistance
  - Provide mechanised assistance to researchers in the formulation of schedulability tests
  - Use evolutionary algorithms to semi-automate deriving response time equations

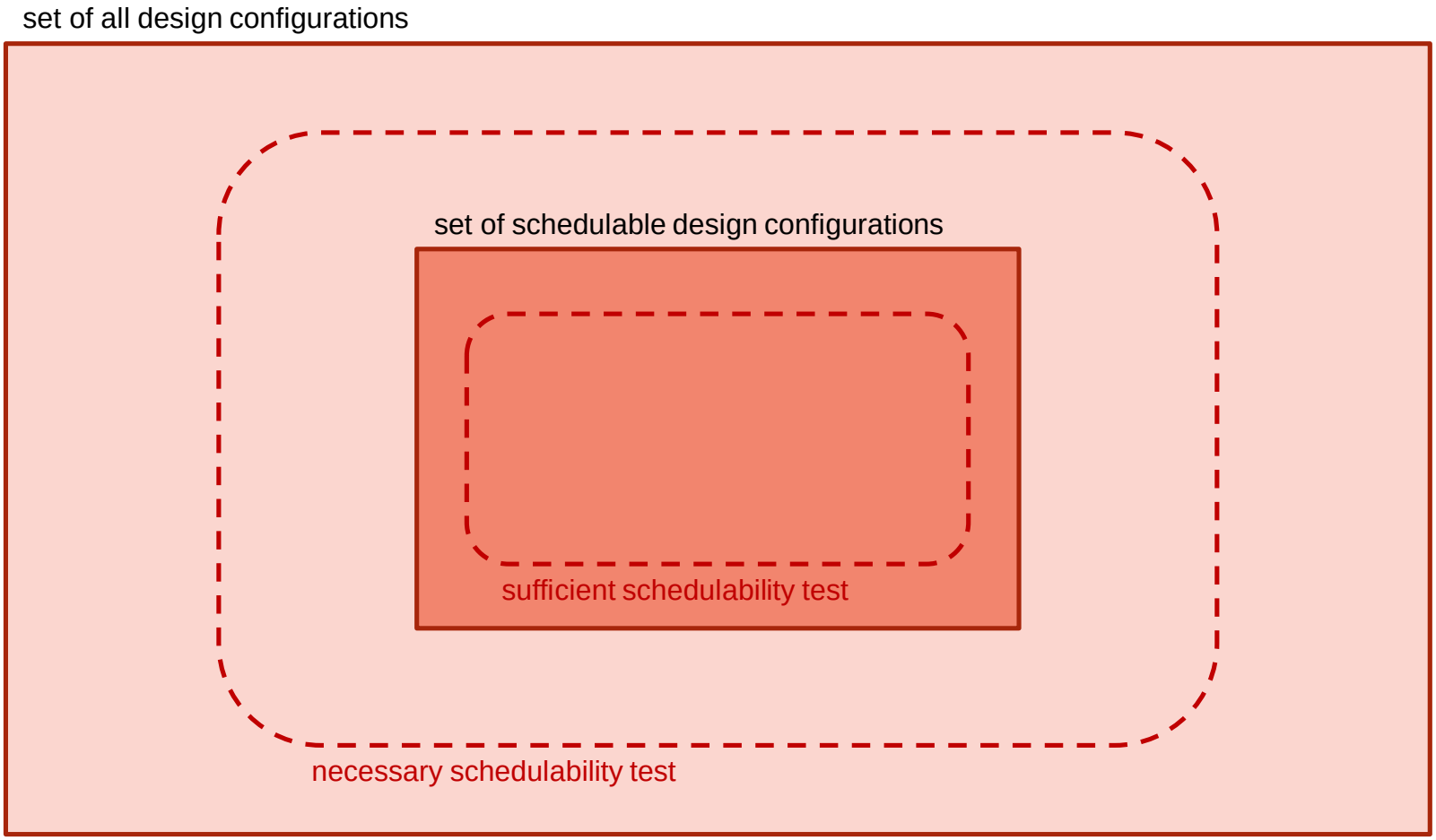
Evolutionary Algorithm

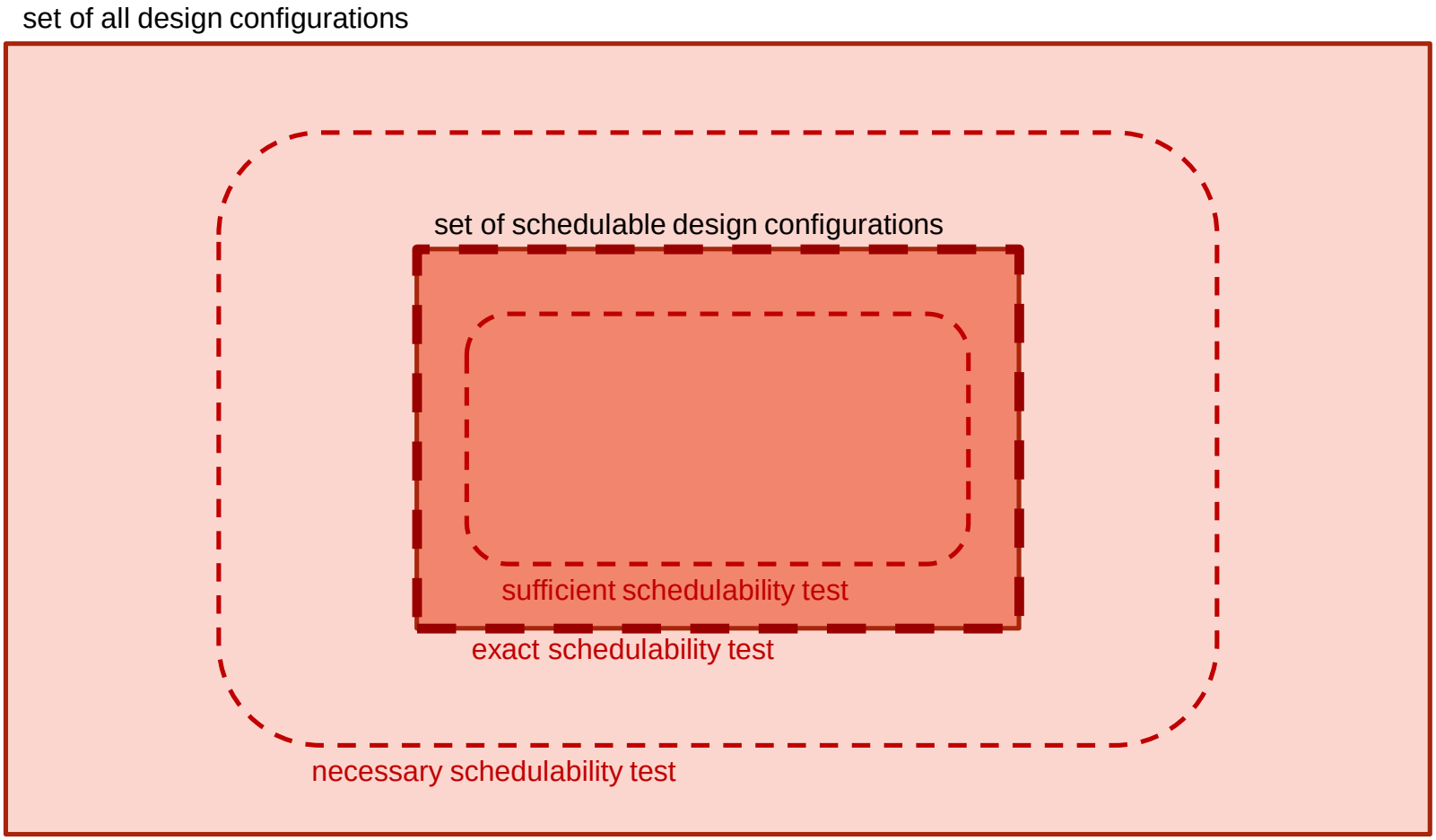


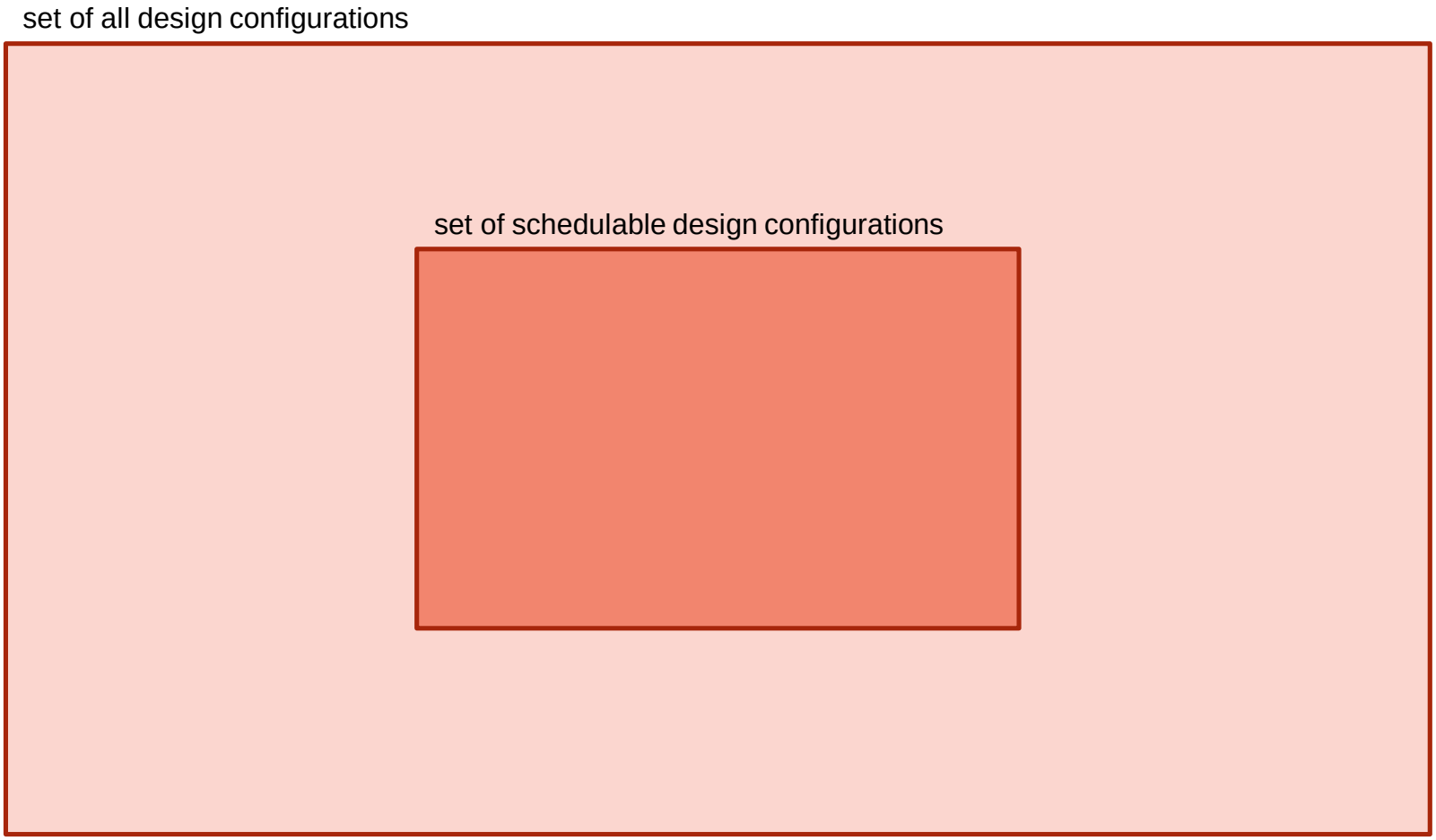
set of all design configurations

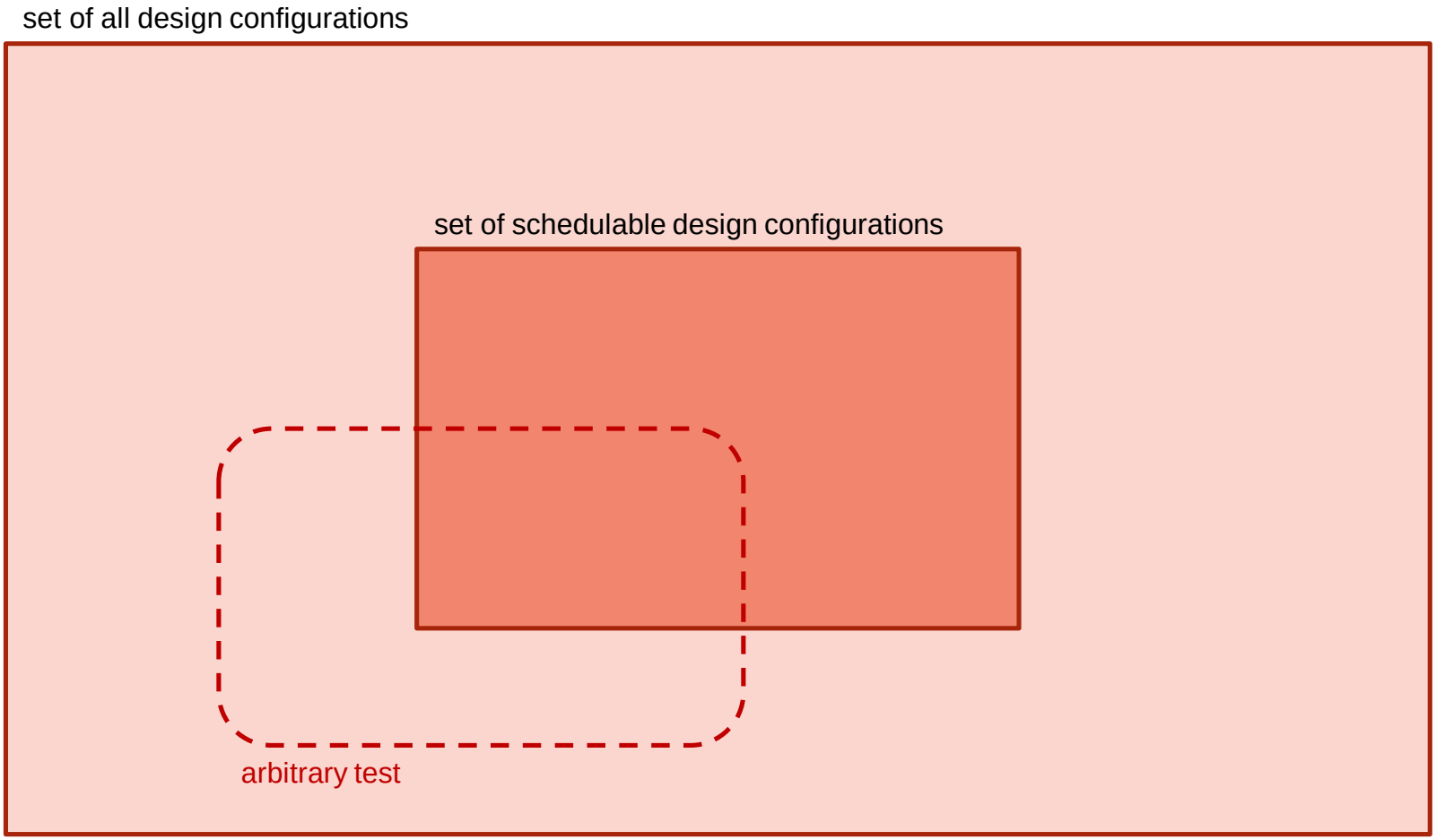

set of all design configurations
set of schedulable design configurations

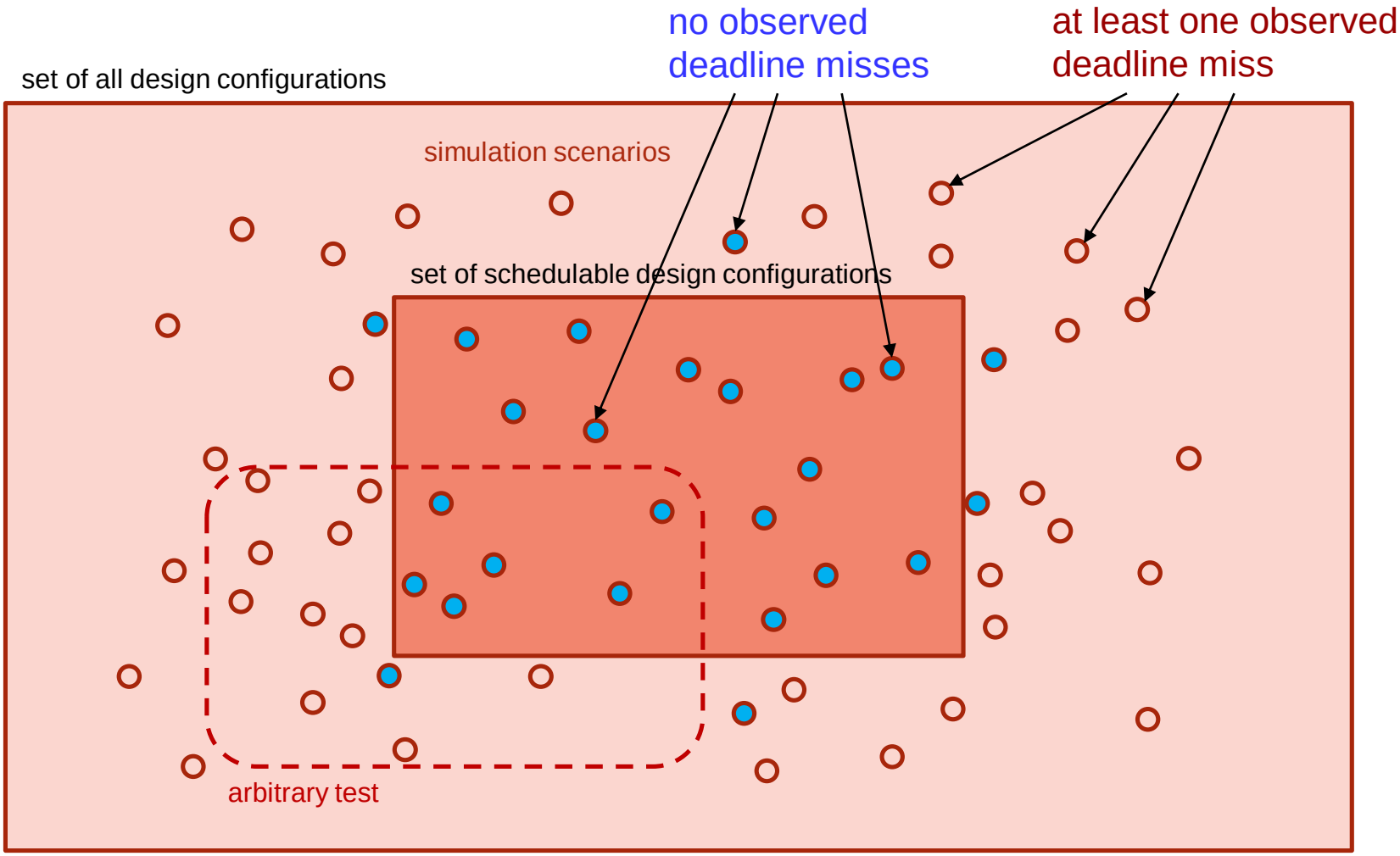

set of all design configurations
set of schedulable design configurations
necessary schedulability test

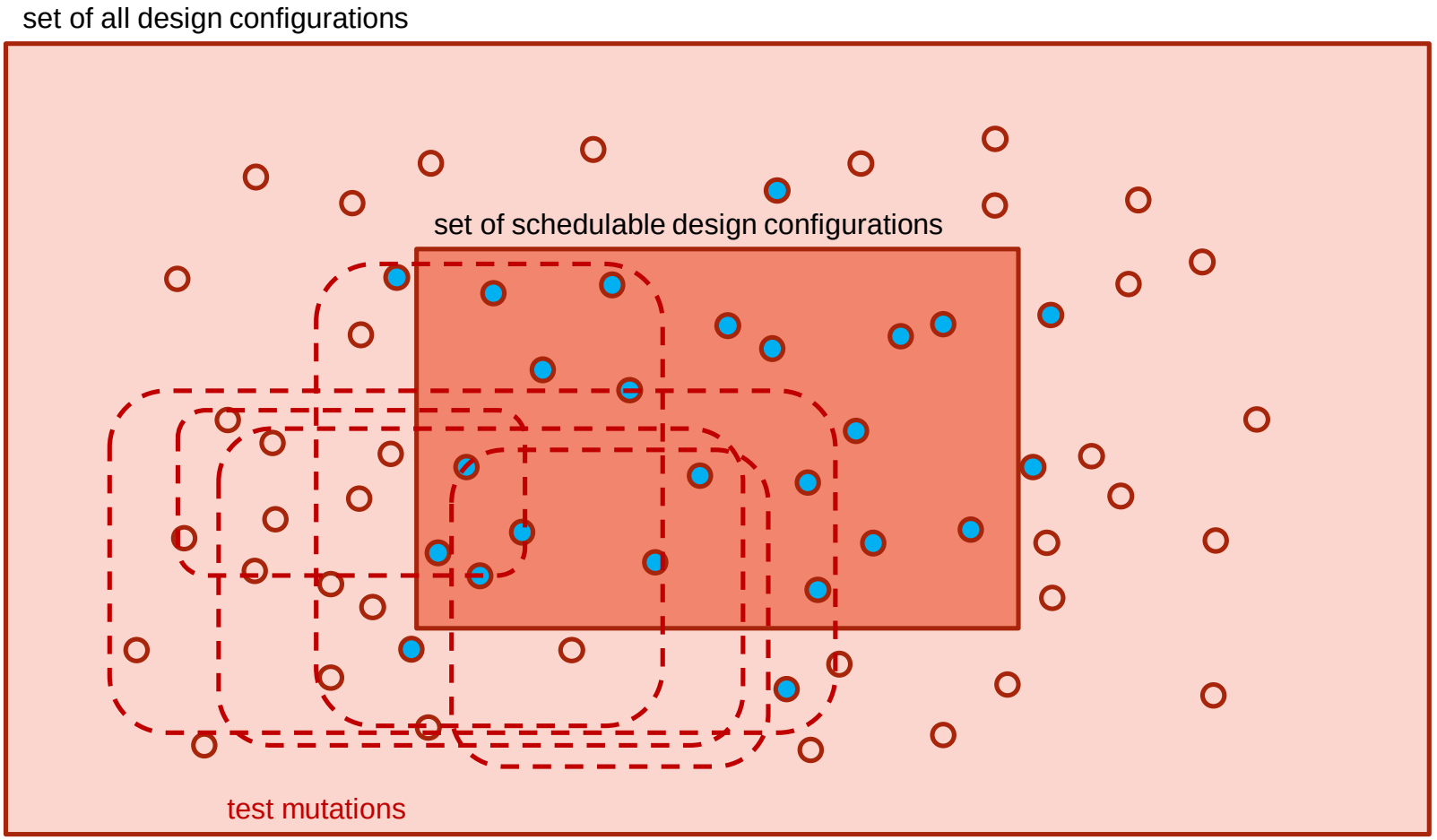

set of all design configurations
set of schedulable design configurations
sufficient schedulability test
necessary schedulability test

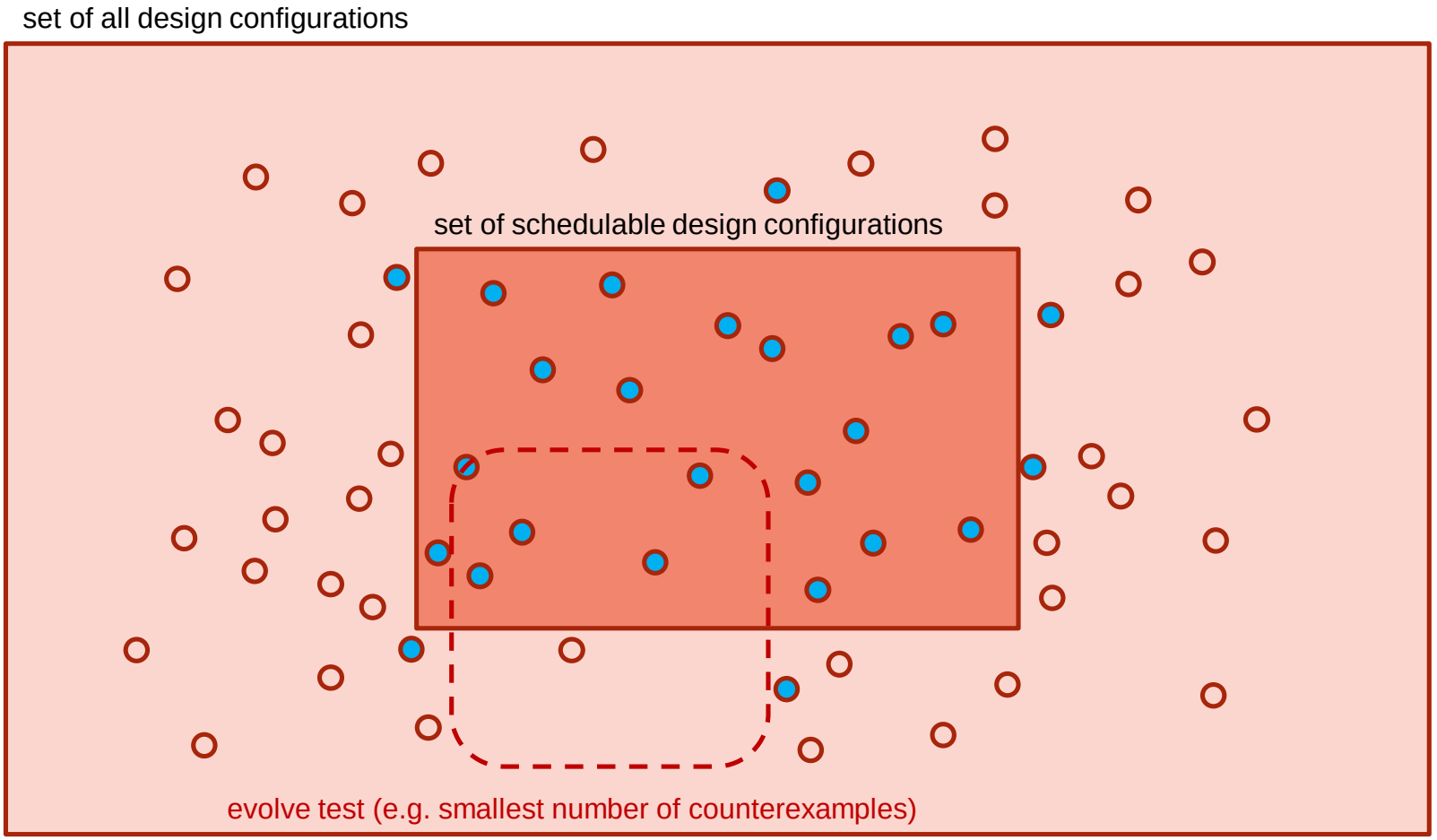

set of all design configurations
set of schedulable design configurations
sufficient schedulability test
exact schedulability test
necessary schedulability test

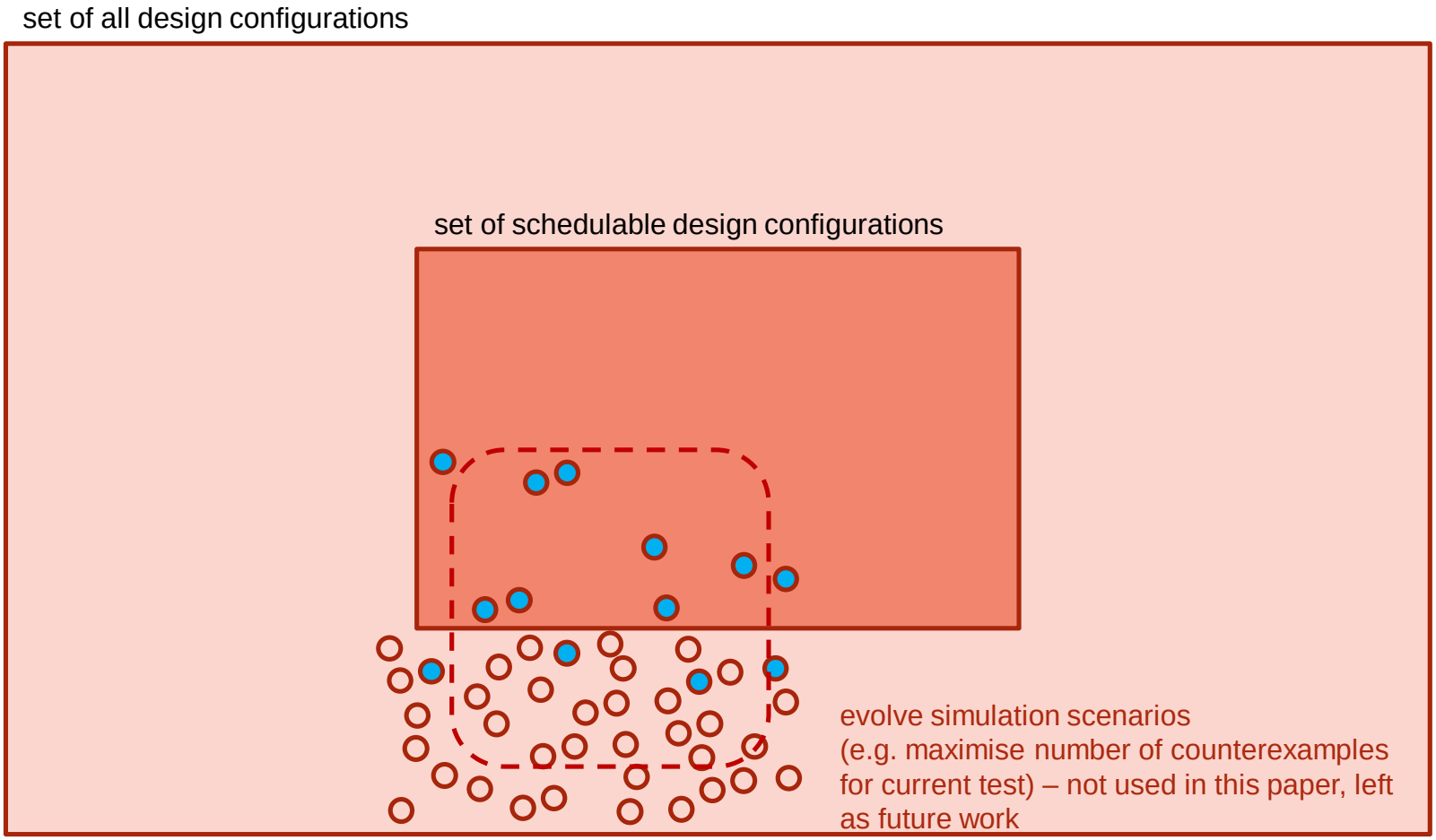

set of all design configurations
set of schedulable design configurations

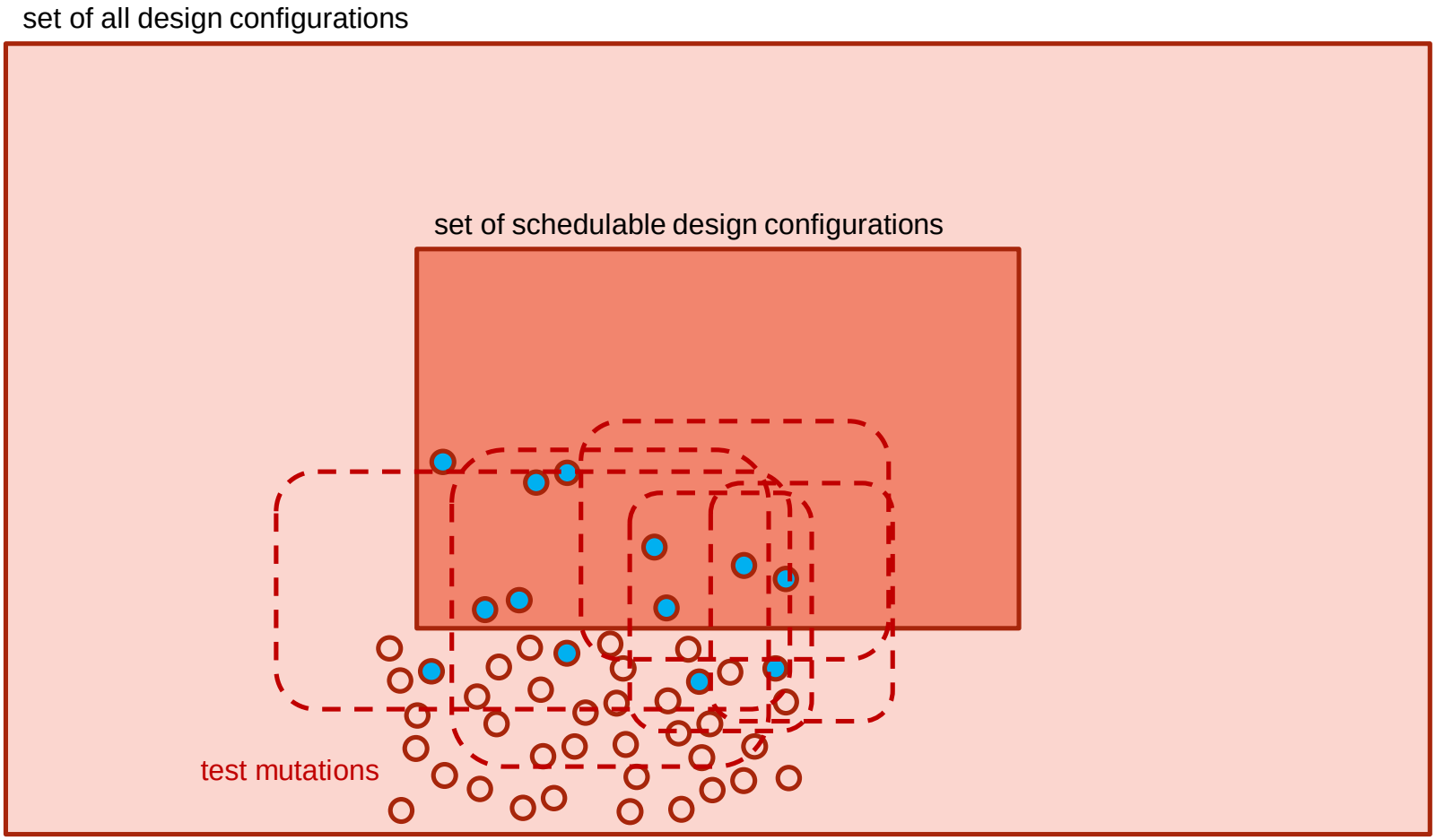

set of all design configurations
set of schedulable design configurations
arbitrary test

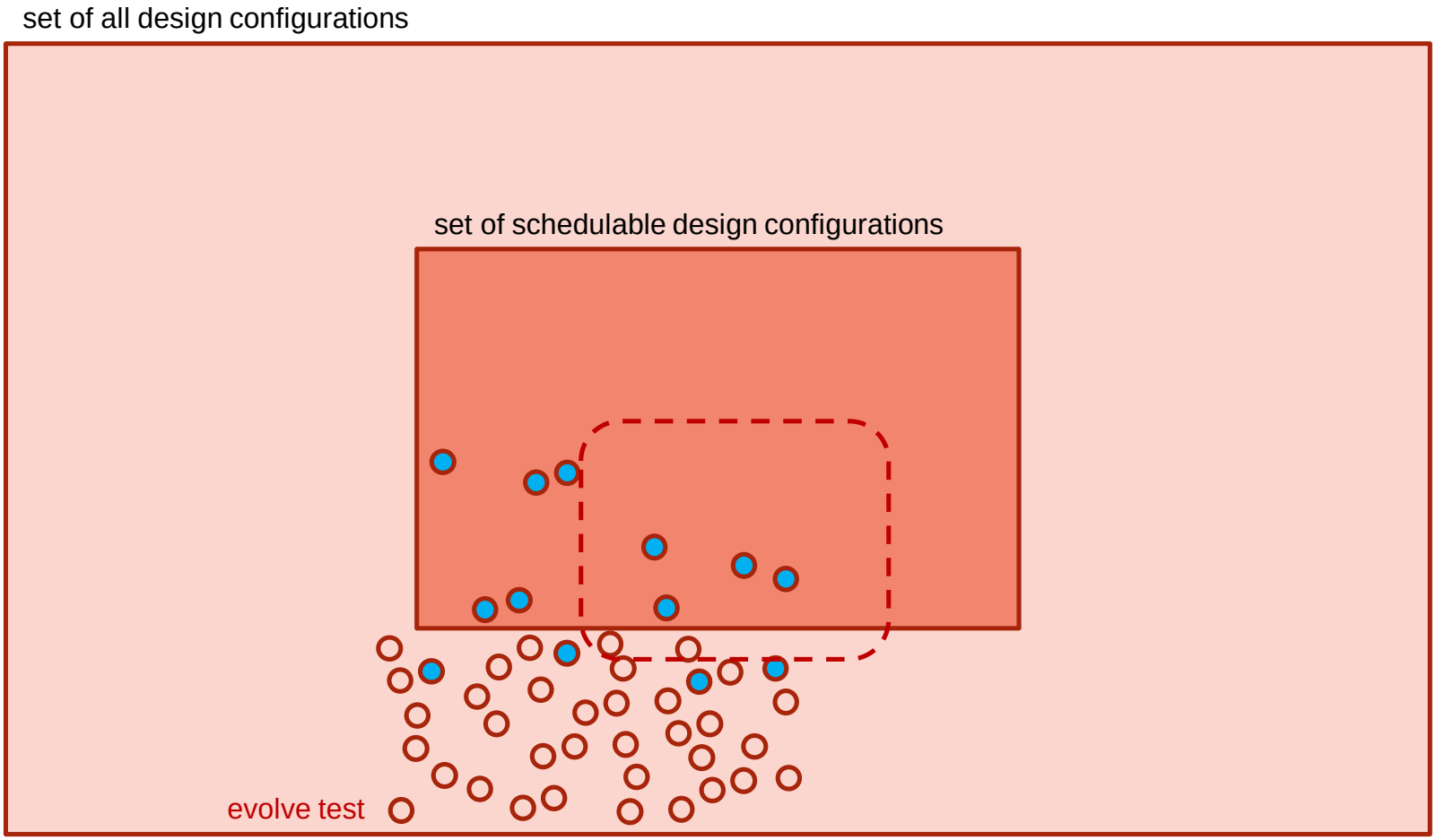

no observed
deadline misses
at least one observed
deadline miss
set of all design configurations
simulation scenarios
set of schedulable design configurations
arbitrary test


set of all design configurations
set of schedulable design configurations
test mutations


set of all design configurations
set of schedulable design configurations
evolve test (e.g. smallest number of counterexamples)


set of all design configurations
set of schedulable design configurations
evolve simulation scenarios
(e.g. maximise number of counterexamples
for current test) – not used in this paper, left
as future work


set of all design configurations
set of schedulable design configurations
test mutations


set of all design configurations
set of schedulable design configurations
evolve test

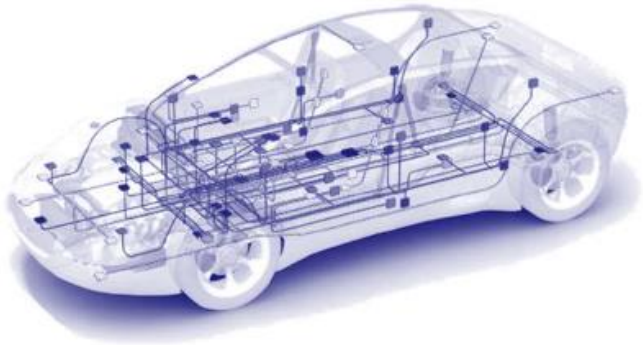
Formulation Assistance in detail

- Researchers
  - Consider the system model and scheduling policy
  - Determine a set of *symbols* and *operators* forming a *grammar* that can be used to express candidate response time analysis equations
    - Examples of symbols: C_i, D_i, T_i, J_i, B_i, R_i
    - Examples of operators: +, -, *, /, ceiling, floor, max, min, sum
  - Obtains a set of *verification vectors*
  - Each verification vector represents a concrete system
    - Includes the parameters for each schedulable entity (task, message etc.) as well as their indicative response times
  - Determine indicative response times
    - Measurements from a real system
    - Cycle accurate simulation
    - Simulation using a high level model

Formulation Assistance in detail

- Grammatical Evolution
  - Introduced in late 1990s by Ryan and O'Neil, similar to Genetic Programming
  - Performs the evolutionary process on candidate equations that are expressions represented by variable length strings encoded according to a grammar defined in Backus-Naur Form (BNF)
  - Strings are evolved via recombination and mutation that respects the specific rules of the grammar
  - Advantage that the grammar provides direct control over how symbols and operators can be combined enabling domain knowledge to be applied
  - Population-based approach supports parallelism
  - Requires a *grammar* and *fitness function* defined by the researcher

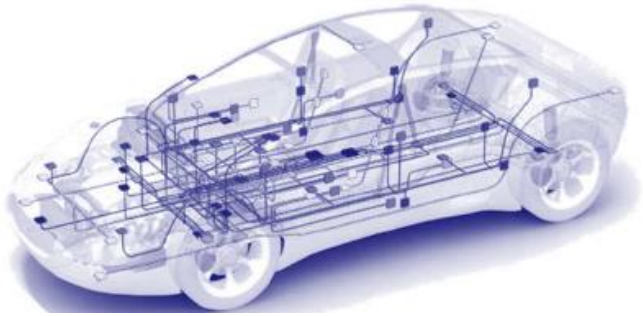
Proof-of-Concept



- Controller Area Network (CAN)
  - Well-known in-vehicle network used in automotive
  - Famously the original analysis was flawed
  - Problems identified and fixed in 2007

    “Controller area network (CAN) schedulability analysis: Refuted, revisited and revised”

Proof-of-Concept



- Controller Area Network (CAN)
  - Exact test E1:
    - Complex two stage test (see paper)
  - Sufficient Test **S1**:

$$R_i = J_i + C_i + max(B_i, C_i) + \sum_{k \in hp(i)} \left(\left\lfloor \frac{R_i - J_i - C_i + J_k}{T_k} \right\rfloor + 1 \right) C_k$$

  - Sufficient Test **S2**:

$$R_i = J_i + C_i + max(B_i, C_i) + \sum_{k \in hp(i)} \left(\left\lfloor \frac{D_i - J_i - C_i + J_k}{T_k} \right\rfloor + 1 \right) C_k$$

  - Flawed Test **F1**:

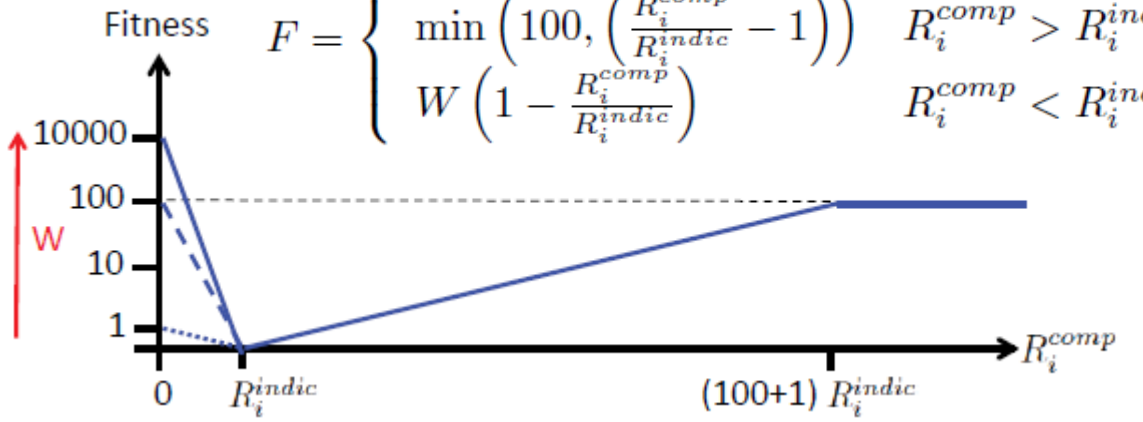
$$R_i = J_i + C_i + B_i + \sum_{k \in hp(i)} \left(\left\lfloor \frac{R_i - J_i - C_i + J_k}{T_k} \right\rfloor + 1 \right) C_k$$

  - Proof of concept aims to evolve sufficient tests of the form R_i = <expr>. (can be recursive like **S1** or non-recursive like **S2**)

Proof-of-Concept: Fitness

- Fitness function
  - Compares response time computed for each message using the candidate expression with an indicative response time stored in the verification vectors
  - Fitness summed for every message in each of 100 verification vectors

$$F = \begin{cases} 0 & R_i^{comp} = R_i^{indic} \\ \min\left(100, \left(\frac{R_i^{comp}}{R_i^{indic}} - 1\right)\right) & R_i^{comp} > R_i^{indic} \\ W\left(1 - \frac{R_i^{comp}}{R_i^{indic}}\right) & R_i^{comp} < R_i^{indic} \end{cases}$$



  - Weighting factor W increases from 1 to 10,000 over generations to penalise under-approximation as evolution progresses

Proof-of-Concept: Grammar

- Dimensionality and Scale Invariance
  - Must ensure that expressions generated have the correct dimensionality (i.e. are in the same units as R_i for example $R_i = D_i$ is dimensionally correct whereas $R_i = D_i * D_i$ is not
  - Also need to ensure scale invariance so that the expressions work correctly even if the measurement scale for all parameters is changed
  - Avoid use of individual ceiling, floor, and multiply operators, instead use compound operators $\lceil A/B \rceil C$ and $\lfloor A/B \rfloor C$ that preserve dimensionality
  - Reduces the size of the search space eliminating all dimensionally incorrect expressions
- Constraints
  - Embed domain knowledge in the grammar to constrain where certain symbols and operators can be used
  - Response times can only be composed of multiples of C_i, C_k, J_i, B_i all other symbols can only be used in their multipliers

Proof-of-Concept: Grammar

<test> ::= R_i = <expr>
<expr>::=
(<expr><op><expr>)|<iVar>|
<expr>*(<exprInFloorCeil>$<exprInFloorCeil>)|
<expr>*(<exprInFloorCeil>/<exprInFloorCeil>)|
sigma(<range>)(<exprInSum>)|((<expr>)~(<expr>))|
((<expr>)_(<expr>))
<exprInSum>::=
(<exprInSum><op><exprInSum>)|<iVar>|<kVar>|
<exprInSum>*(<exprInSumFloorCeil>$<exprInSumFloorCeil>)|
<exprInSum>*(<exprInSumFloorCeil>/<exprInSumFloorCeil>)|
((<exprInSum>)~(<exprInSum>))|
((<exprInSum>)_(<exprInSum>))
<exprInFloorCeil>::=
(<exprInFloorCeil><op><exprInFloorCeil>)|<iVar>|
<iNumDenVar>|sigma(<range>)(<exprInSumFloorCeil>)|
((<exprInFloorCeil>)~(<exprInFloorCeil>))|
((<exprInFloorCeil>)_(<exprInFloorCeil>))
<exprInSumFloorCeil>::=
(<exprInSumFloorCeil><op><exprInSumFloorCeil>)|
<iVar>|<kVar>|<iNumDenVar>|<kNumDenVar>|
((<exprInSumFloorCeil>)~(<exprInSumFloorCeil>))|
((<exprInSumFloorCeil>)_(<exprInSumFloorCeil>))
<op>::=+|-
<range>::=forall_k_InLp_i|forall_k_InLep_i|forall_k_InHp_i|
forall_k_InHep_i|forall_k
<iVar>::=B_i|C_i|J_i
<kVar>::=C_k
<iNumDenVar>::=T_i|D_i (or in the recursive case ::=T_i|R_i)
<kNumDenVar>::=T_k|J_k

Proof-of-Concept:

- Evaluating recursive equations
  - Start with the no interference response time $J_i + C_i$
  - Repeatedly evaluate until the result converges
    - Or a max number of iterations is reached (100 in our experiments)
  - If result causes a divide-by-zero or overflow, then stop and use 0 or a large integer for the resulting value (results in poor fitness)

Proof-of-Concept: Experiments

- Verification vectors
  - Assume CAN bus with 11-bit message IDs
    - Messages from 1 to 8 data bytes
    - Period log-uniform in 5 to 500ms
    - Deadline in range 0.5 to 1.0 times period
    - Release jitter in range 0 to 0.5 times deadline
  - 100 verification vectors of 20 messages each with priorities in Deadline Monotonic order, utilisation from 50% to 97.5% (43 schedulable)
  - 25 verification vectors of 20 messages each in random priority order, utilisation from 25% to 35% (7 schedulable)
  - 10 verification vectors of 10 messages that highlight the flaw in the original CAN analysis, priorities in Deadline Monotonic order, utilisation from 95% to 99.5% (all unschedulable)

Proof-of-Concept: Experiments

- Grammatical Evolution
  - Implemented in EpochX open source genetic programming framework (v1.4.1)
    - Population size 1000
    - Generations 2000
    - Mutation rate 0.1
    - Fitness proportionate selection
  - Each experiment was repeated 500 times
- Assessment of evolved schedulability tests
  - Against assessment vectors (similar to verification vectors but 10 times as many message sets)
  - Manual checking for sufficiency (or not) via proof sketches and counterexamples

Proof-of-Concept: Experiments

- Expt 1: Baseline
  - No recursion allowed by the grammar, precise indicative response times used, no corner case verification vectors

```
((Ji+sigma_(forall_k_InHp_i)
(((Ci)~(Ck*(((Tk+(Ci+(Jk-(Ji-Di))))-Tk)$Tk)))))+(Bi+Ci))
```

$$R_i = J_i + C_i + B_i + \sum_{k \in hp(i)} \max\left(C_i, \left\lceil \frac{D_i - J_i + C_i + J_k}{T_k} \right\rceil C_k\right)$$

  - Fitness 12624 (comparable to test **S2** – fitness 12038)
  - Proven sufficient (sketch proof in appendix of paper)

Proof-of-Concept: Experiments

- Expt 2: Adding recursion
  - Recursion allowed by the grammar, precise indicative response times used, no corner case verification vectors

```
 ((((Ji+(Bi+(Ci+sigma_(forall_k_InHp_i)(Ck)))))~(Bi))
+sigma_(forall_k_InHp_i)(Ck*(((Ri-(Ji-Jk))
-((((Tk)~(Ck)))~(((Jk)~((Bi+((Tk-Ji)-Ck)))))))$Tk)))
```

$$R_i = J_i + C_i + B_i + \sum_{k \in hp(i)} \left\lceil \frac{R_i - J_i + J_k - \max(0, B_i - J_i - C_k)}{T_k} \right\rceil C_k$$

  - Fitness 3084 (improves over test **S2** – fitness 12038, but still some way off test **S1** – fitness 1702)
  - Proven not sufficient (counter example in appendix of paper)
    - Can sometimes produce optimistic results when $B_i - C_k > C_i - \tau_{bit}$
      i.e. when the blocking factor is more than the transmission time of the message or interest plus that of a higher priority message

Proof-of-Concept: Experiments

- Expt 3: Adding Corner Cases
  - Recursion allowed by the grammar, precise indicative response times used, corner case verification vectors highlighting the flaw in test **F1**

```
(Bi+(Ci+(Ji+sigma_(forall_k_InHp_i)
(Ck*(((((((Ri)_(Tk)))~(Ck)))_(Jk))+(Ri-Ji))$Tk)))))
```

$$R_i = J_i + C_i + B_i + \sum_{k \in hp(i)} \left\lceil \frac{R_i - J_i + \min(\max(R_i, C_k), J_k)}{T_k} \right\rceil C_k$$

  - Fitness 3096 (similar to Expt 2, but still some way off test **S1** – fitness 1702)
  - Proven not sufficient (counter example in appendix of paper)
    - Can sometimes produce optimistic results when the release jitter of a higher priority message is very large permitting back-to-back interference (case not covered in the verification vectors)

Proof-of-Concept: Experiments

- Expt 4: Adding Corner Cases
  - Recursion allowed by the grammar, approximate indicative response times used from 0.8 to 1.0 times the exact values, corner case verification vectors highlighting the flaw in test **F1**

```
((Ji+(Bi+Ci))+sigma_(forall_k_InHp_i)
(Ck*((((Ci+Ri)+Jk)-Ji)$Tk)))
```

$$R_i = J_i + C_i + B_i + \sum_{k \in hp(i)} \left\lceil \frac{R_i - J_i + C_i + J_k}{T_k} \right\rceil C_k$$

  - Fitness 4710 (somewhat worse than Expts 2 and 3)
  - Proven sufficient (sketch proof in appendix of paper)

Proof-of-Concept: Experiments

- Expt 5: Random search
  - Recursion allowed by the grammar, exact indicative response times, corner case verification vectors highlighting the flaw in test **F1**
  - Search not directed just tries different expressions at random and returns the one with the best fitness

```
((((((((((Ji)~(Bi)))~(Bi*(Ci$Ri))))~(Ci*(Ti$Ci))))~(Ji)))~
(Ji))+sigma_(forall_k_InHp_i)(Ck))
```

$$R_i = \left\lceil \frac{T_i}{C_i} \right\rceil C_i + \sum_{k \in hp(i)} C_k$$

  - Fitness 56477 (an order of magnitude worse than the evolutionary approach)
  - Typically returns values greater than period so nearly all messages deemed unschedulable (useless as a schedulability test)

Lessons learnt #1

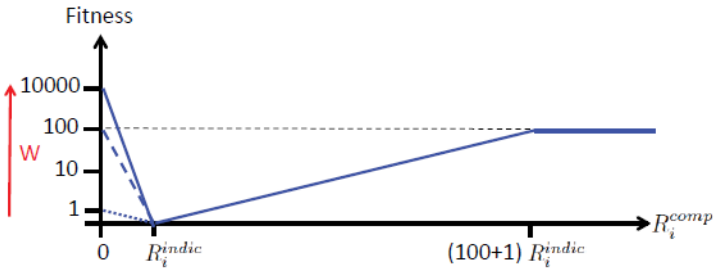
- Improving the grammar
  - **Dimensionality**:
    - Restricted use of ceiling, floor, and multiply to only appear in compound operators – ensures expressions are scale invariant and dimensionally correct
  - **Nesting restrictions**:
    - Constrained summation so it could not be nested – avoids very complex expressions
  - **Symbol restrictions**:
    - Actual response times can only be composed of multiples of C_i, C_k, J_i, B_i all other symbols can only be used in their multipliers

Lessons learnt #2

- Improving the verification vectors
  - **Correlations between parameters**
    - Had to be avoided as this could lead to inappropriate substitution of one parameter for another e.g. if deadlines were in DM order
  - **Range of values for variables**:
    - Small values for release jitter initially used meant that J_i could serve as an incorrect proxy for C_i or B_i
    - Expression from Expt 3 is not sufficient when jitter values are very large – points to need to improve the set of verification vectors
    - Transmission times had to take a range of values (not all the same) otherwise C_i, J_i, and B_i would not be distinguished - similarly if implicit deadlines were used D_i, and T_i could not be distinguished

Lessons learnt #3

- Tuning the Grammatical Evolution
  - **Fitness function**
    - Vitally important needs to heavily penalise under-approximation of response times, but a heavy penalty in early generations can be counter productive hence the variable weighting factor used



  - **Number of generations**
    - Explored 500, 1000, 2000, 5000 no significant improvement after 2000
  - **Escaping local minima**
    - Grammatical Evolution is a trade off between exploration of the vast search space and exploitation of good candidates
    - Small changes to an expression can result in large changes to fitness that cause difficulties in escaping local minima
    - Mitigated by repeating experiments a large number of times (500) and taking the best overall results

Lessons learnt #4

- Improving the implementation
  - **Parser**
    - Standard EpochX parser is capable of handling arbitrary expressions, which makes it slow when all that is needed is to parse mathematical expressions
    - Implementing our own parser (two pages of Java code) improved the overall runtime by a factor of 100
  - **Parallel execution**
    - Using a high performance compute cluster enabled evolving the 500 populations for each of our experiments in parallel

Conclusions

- Research Contributions from the Proof-of-Concept
  - **Formulation assistance** using Grammatical Evolution is a viable approach that can provide interesting candidate equations for schedulability tests
  - **Candidate equation quality** similar to known sufficient tests in terms of performance w.r.t. the assessment vectors
  - **The burden of proof (of sufficiency) remains with the researcher**
  - Formulation assistance is only assistance **not** automation
    - Of the four experiments the best results from two were sufficient tests, while the other two were not
  - Some performance degradation when approximated indicative response times are used (as would be the case when no exact test is known and the worst-case cannot be simulated) but still resulted in useful schedulability tests

Future Research

- Directions
  - Improving the basic method
    - Explore an **Island Based Approach** that enables re-combination and evolution of the best solutions found in different island populations evolved in parallel – aims to avoid evolutionary dead-ends (local minima)
  - Unsolved schedulability analysis problems
    - Apply the formulation assistant concept to unsolved schedulability analysis problems, in the area of Network-on-Chip, using simulation as a means of determining indicative response times
  - Co-evolution of verification vectors
    - Important to avoid candidate equations being wrongly classified as sufficient – this is difficult as simulation can only provide necessary tests
    - Possible improvement is to co-evolve verification vectors to better identify corner cases
- Formulation assistance not automation
  - Researchers still need to interpret the expressions derived, refine them and seek to prove their correctness by other means (e.g. proof assistance)

UNIVERSITY of York

Synthesizing Real-Time Schedulability Tests using Evolutionary Algorithms: A Proof of Concept

Piotr Dziurzanski, Robert I. Davis, Leandro Soares Indrusiak

Real-Time Systems Group
Department of Computer Science
University of York
United Kingdom

RTS York