Chris Cargile

2.2) Specification Languages (SL) describes a system diagrammatically, in terms of:
* Nodes => functions (nodes B,C might nest in ‘A’, for example);
* Arcs => data flows; and

Checks for self-consistency might include:

--> do the functions (which ===data transformation) always maintain the same invariant(s) and when nesting is illustrated, do the nested transformations facilitate/permit adhering to invariant of nestee? (ie: functionA(int a , int b) { return (<T>double)?--functionB (a , b) { return double } }

--> Are null flows allowed/disallowed consistently?

--> Can functions requiring/forbidding data/null receive their counterparts consistently OR shows ‘throwing a flag’ consistently?

--> Are recursively written functions portrayed the same, always

2.4) a) programmer implements coarse-grain locking, preventing interleaving of accesses at trade-off of unnecessary blocking b/n threads:
* this mechanism of concurrency control introduces pessimistic inaccuracy as some gains that may have been possible may not be realized, so the analysis is certainly safe, but it is not conservative at all

b) automated static analysis verifies serializability with finer-grain locking, when some methods don't even use locks:
* as this mechanism can still reject valid sets of methods that would ensure serializability, it introduces pessimistic inaccuracy.

c) programmer required to use a particular concurrency control protocol and static analysis checking conformance to protocol:
* statically checking the adherence to a specific protocol in a simplified property analysis means.

d) augmenting the data accesses to build a serializability graph structure representing the 'happens before' relation among transactions in testing:
* checking the possibility of cycles in a graph is an analysis that can be performed resolutely, and is a safe, sound, and conservative type of analysis, therefore this type of analysis is described under the simplified property genre.