Applicability of Object-Oriented Invariants to the *HazGas* System

Group 10

Our system, *HazGas*, in general, lends itself well to object-oriented programming – there exist "rooms" and "controllers" which can easily be modeled as objects. However, it has been modeled using *Spin*[1] – a model checker used to verify multithreaded programs which uses the concept of *processes* to model interactions between concurrent aspects of the system. As such, object invariants are not directly applicable to our specific implementation as Spin is not an object-oriented language and takes a more imperative approach to systems programming.

If we were to implement this with objects, we could have *unpack* and *pack* directives which would be executed before and after, respectively, modifications of room and controller state (gas level, venting status, etc.). This enables us to ensure that certain invariants are not violated, e.g. room is venting if the upper gas threshold is exceeded. This means that invariants can only be violated when in an "invalid" state (as defined by the paper) and therefore we can check the correctness of the system if all objects within it are in the "valid" state.

Spin, however, enables the use of LTL to verify invariants over the entire model. This is comparable to unpack/pack as these LTL expressions can be used to indicate pre- and post-conditions for a given invariant to hold, e.g. "when upper gas threshold is exceeded (pre-condition), the room is venting (invariant) until the gas volume is below the lower gas threshold and the factory is not alarming (post-condition)". This, however, is not as precise as object invariants as there is not a specific time frame in which the invariant must hold – whereas with object-oriented invariants, we can specify the precise location where the object must be packed and validated.

Packing can be simulated with the use of *assert* statements in appropriate locations – all the invariants must be atomically asserted and then, if the assertions hold, the state can be assumed to be valid. We have chosen not to take this approach, as it does not allow us to verify a significant new set of requirements and LTL is sufficient to ensure the various guarantees we want to verify about our system.

[1] http://spinroot.com/spin/what.html