JML Workshop 2016

Specification Language Semantics and Use Cases

Haus der Kirche, Bad Herrenalb, November 2–4, 2016

Minutes

version: 2016-11-30.

1 Combination of Approaches and Tools

- Nikolai's presentation showed how combination of different tools and technique within Frama-C could be used in unexpected ways, for example to make an automatic suggestion about unprovability of a specification.
- Inspired by Nikolai's presentation of Frama-C, we discussed how tools can build on a central infrastructure.
- Have a common parser/AST data structure (keep up with JDK advances)
- David Cok's work (with John Singleton and others) on OpenJML builds on OpenJML's AST already (works on invariant inference from ASTs or basic blocks).
- Worth thinking about: A "give-and-take" communication would be interesting in which tools can report what results they have produced.
- We should encourage interoperability of tools.
- We need a common understanding of semantics.

2 Formal Semantics

- Formalising a semantics in Coq/Isabelle/PVS would add to the strength of the presentation; make it more rigorous.
- Semantics should also be available in "mathematics" for people not working with that prover.
- During work on the semantics, we should think about potential extension to concurrency (atomicity etc)
- The notion of modularity was central in the discussion of Ulbrich's semantics. We agreed it needs to be refined. We identified at least three levels of modularity:

  1. no modularity (closed program analysis)
  2. lib-modularity (new classes can be added without invalidating existing proofs or conclusions; class-modularity as special case of that)
  3. method-modularity

  It may be worth looking at the 2015 *TOPLAS* paper by Leavens and Naumann, which has a definition of modular reasoning.
- Example: Is this program correct? (Disregarding overflow for now!)

```
class C {
    private int x = 0;
    //@ ensures \result > 0;
    int m() {
        return ++x;
    }
}
```

  It is no-mod correct, it is lib-mod correct, but it is not method-lib correct.
- Care about all states that satisfy a precondition or only only about reachable states?
- Idea: Java semantics should be factored out by referencing to the Java language specification. (Black box idea). Is this always possible?

- Discussion on dealing with exceptions in specifications. The idea of an explicit error value in the semantics (not visible to the user!) does not seem very attractive.
- Instead: Have a partial definition and add well-definedness assertions that ensure that every evaluation remains within these bounds.
- For the semantics definition, define how all JML-constructs are mapped to assumes/asserts interleaved with statements. (Is this always possible?)
- Open issue: object creation within specs (see Daniel's thesis)

3 Breakout Discussions

We split into 4 groups, each discussing a subject and then had a plenary discussion on each of them.

3.1 Compliance levels for programs

Identified items to be checked: – contract feasibility / consistency – deadlocks – liveness – starvation – progress/responsiveness – termiantion – runtime (uncaught) exception freedom – no dead code – absence of "bad" races – security – resource bounds – contract satisfaction – public version private specifications (later called specifications versus annotations)

Levels of granularity: program > module > class > method
(A program consistent on method level is also consistent on class level, etc., but not the other way around).

Discussion:

- behavioral subtyping – does it always exist in JML? Yes, although some expressed doubts.
- termination is not always good – example server, but you can't say what the server does, without looking at the parts of the server which terminate, so you do have to consider termination
- what about a method that satisfies its own contract, but not its invariant?
- one should distinguish between "wrong and wrong", too much of the specification is currently interwined with auxiliary specification needed to help with the proof. Would there be a way to separate the two?
- there is part of the spec that you consider documentation and part that you consider only internally, we are now mixing them
- any written specification ought to be satisfied – but is every annotation a specification? James: "if you have lots of assertions in your program and say this is my specification, I would laugh at you."
- problem is now that we don't distinguish between external specification and internal specificaton (maybe annotations inside the code are not part of the specs)
- is JML a specification language or an annotation language – we did not answer that.

3.2 Semantics of Quantifiers

The teaser was: Is $[\![$`(\exists Integer x; x.value == 42)`$]\!] = [\![$`true`$]\!]$ valid? (1)

- Another example (2): Every person object has a unique number
- do these quantified properties (1),(2) hold for all objects that have been created and initialized until now or about objects that can potentially be created?
- for persons example it may seem natural to use all objects that have been created; for Integer example the other version may be more natural
- we agreed that there are sensible instances for both versions
- why quantified over class type instead of primitive (Integer/int)?
- $\rightarrow$ then quantify over Strings (which are special)
- `List` is also very similar, when you say "forall List" you can think about all Lists you can create.

- maybe what you really want is mathematical datatypes instead all possible objects.
- there was no dispute when it came to `int`s, only about reference types
- for immutable reference types, second, platonic view is more appropriate
- what does exists mean for plationic view? There exists a seven?? that makes no sense!!
- When you implement a parser you might want to specify that there exists a string which has a property...
- you can create additional objects, but there is not point where all objects have been created
- when we do modular verification we do not know what objects have been created at any point
- this has not been solved yet, but it should be in the future!
- AGREEMENT: In any given state the quantifer refers to the objects existing in that state, but it could be a hypothetical state

3.3 Mathematical Datatypes

- there have been modelling types in JML for some time, but they are Java types
- we should use pure mathematical types, in KeY we have the sequence data types, but there should be an extendable framework where we can add data types
- this is how mattias and daniel proposed to do such a framework
- in the group we agreed that we need such types in order to specify thing in a java independent way
- these data types should be heap independent!!
- Not clear: Do we want a Java-similar notation to make datatypes accessible for software engineers or do we want a deliberately different notation to avoid confusion.
- additional thing: add annotations for Java class definitions which are immutable datatypes – that is a similar but not the same: the idea is that we can create an ADT definition from an annotation to an immutable type. Equality is interesting then
- go back to person, maybe person has a variety of fields, whose values is immutable, you will want to use it as a mathematical datatype, but you can't find this in the catalog of mathematical data types
- what does equality mean for such a type? – equality is equality, there is only one empty sequence – when are two non-empty values equal?
- we should look into VeriFast to see what Bart did there
- Wojciech and Mattias introduced two_state and no_state methods...
- this reverses an important design decision of JML, since translating java types to mathematical types, is good for verification but bad for the programmer, since they need to learn more.

3.4 Welldefinedness

- booleans in Java can only have two values
- booleans in JML might be true, false and bottom (exception/undefined/...)
- model this using
  1. undetermined total function
  2. three valued
  3. dodging the issue, by giving only partial, not total definition
- in a paper it was shown that static checking was inconsistent with runtime checking
- if universal quantifer and one of the quantified things throws an exception, does the whole thing throw an exception?!?
- we have anomalies, but in many cases they help us with proving
- third solution you assume averything is welldefined so maybe not so much problem

- we use it in KeY, Michael implemented it
- in Frama-C it is different for runtime (2 or 3) and static (1) verification – 1/0 == 1/0 you can be proven in Frama-C statically but will be undefined at runtime checking
- NaN == NaN false, NaN != NaN also false!!!
- there should be different types of exceptions for non-welldefinedness
- assert 1/0 should be the same as assert if – a parser error but not the same as assert false.

4 Industry

see slides put together by James Hunt.

5 New features

see slides put together by David Cok.

6 Open discussion points

Floating Points

- `x==x` is not necessarily true for floats,
- Wolfgang suggested `\fp_nan(x)` as indicator for floats that are "not a number"
- Can be replaced by `Float.isNaN(x)` or `x!=x`; no need to extend syntax.
- Frama-C: nan are implicitly excluded

Method invocations in object invariants

- The problem already discussed in Leiden: A method call requires the objects precondition, so using method calls in invariants leads to a ill-founded recursion.
- Methods can be annotated `helper` and then used in invariants.
- `helper` does not seem to transport the right intuition here.
- Formal semantics for method calls in specs to be defined: the contract or the actual return value of the call?

Default modifier in interfaces

- current situation: in interface declarations, any ghost/model field declaration, any invariant, any constraint clause is `static` by default, must be declard using JML-modifier `instance` if meant as non-static member. Model methods are (like methods) `instance` by default.
- Rationale is: Field declarations in interfaces are always automatically `static`; instance field cannot be declared.
- But does this automatically extend to invariants?
- In JML there is more possibilities: instance does make sense and seems to be the more frequent use case.
- Experience by many people present shows: This is counter-intuitive. Authors of a tutorial noticed missing `instance` through tool usage.
- General sentiment to remove this default.

7 Framing in JML

- Yuyan presented peaceful coexistence of separation logic and region logic.
- Region logic is semantically nearly identical to dynamic frames in KeY.
- Experiences with KeY: Disjointness needs to be explicitly mentioned, numerous proofs are "expensive".
- The combination could be run on top of KeY if separation logic is encoded using regions, could be run on top of viper if regions are (somehow) encoded into separation logic.
- Having possibility to mix separation logic and dynamic frames specifications:
  - very expressive spec tool
  - user does not want to experience two framing paradigms in one program
  - long term goal: user should not need to worry about framing at all.
  - needed for research purposes to be able to experiment; flexibility is most important.
- semantics of invariants / abstraction predicates bound to specification technique
- The idea of David's `@Secret` annotations for observably pure methods could be encoded here.
- Keep concurrency in mind!!! (There is a permission implementation for JML in KeY by Wojciech.)

8 A book on JML

- David Cok is writing a book on OpenJML – this is independent of other book enterprises. (tutorial, guide to OpenJML, reference)
- Possible structure of a book on JML:
  1. Introduction
  2. Syntax and semantics
  3. tool usage
  4. open call for research papers and
  5. future work
- The initiative did not really enjoy much endorsement
- Rather: Have a collection of important documents on a well-equipped modern webpage:
  - Precise language definition: syntax, semantics
  - Tutorials (e.g. the one in KeY book)
  - Tool tutorials
  - Short videos
  - ...
- **Instead: Make an effort to standardise JML** with a good process
- JML Standardization Committee
  - Gary Leavens
  - Mattias Ulbrich
  - David Cok
  - Erik Poll
  - James Hunt
- First Goal: Identify the core language which is to be part of the standard. — Have this done by March 17, 2017.
- John Singleton and Malte Schwerhoff will look at a good infrastructure to make program verification examples easily accessible. There already exists a lot on the web, but we need a way to make it easy to find it. Marieke provides them with links of known collections.

Minutes by Mattias Ulbrich `<mulbrich@kit.edu>`
notes taken by Michael Kirsten and Mihai Herda.