Proving safety of SupraBFT using Microsoft Ivy

Chandradeep Dey*

under supervision of Prof. M Praveen

Chennai Mathematical Institute

*with contributions from G Namratha Reddy

Formal methods in protocol verification

- Formal methods provide guarantees about protocols
- Faithful implementations avoid entire classes of errors

Logic

- Guarantees need to be specified in some logic
- Classical result – First order logic is undecidable (Church, 1936)

Logic – Decidable fragments

- Bernays-Schönfinkel-Ramsey class
  - Also called the effectively propositional reasoning (EPR) fragment
  - $\exists^*\forall^*$ in prenex normal form
  - No function symbols
  - Decidable
    - Skolemisation to replace $\exists$
    - Replace $\forall$ with all possible values using only constant symbols
  - Not very expressive

Logic – Decidable fragments

- Bernays-Schönfinkel-Ramsey class
  - Also called the effectively propositional reasoning (EPR) fragment
  - $\exists^*\forall^*$ in prenex normal form
  - No function symbols
  - Decidable
    - Skolemisation to replace $\exists$
    - Replace $\forall$ with all possible values using only constant symbols
  - Not very expressive ☹

Logic – Decidable fragments

- Bernays-Schönfinkel-Ramsey class
- Extending the fragment
  - Multiple sorts
  - Function symbols are back

Logic – Decidable fragments

- Bernays-Schönfinkel-Ramsey class
- Extending the fragment
  - Multiple sorts
  - Function symbols are back – but stratified
  - Finite relevant vocabularies (Ge and de Moura, 2009)
    - Sets of ground terms corresponding to universally quantified variables, uninterpreted functions, and uninterpreted relations, that are enough to check satisfiability

Logic – Decidable fragments

- Bernays-Schönfinkel-Ramsey class
- Extending the fragment
- The finite essentially uninterpreted (FEU) fragment
  - Allow interpreted formulae
  - Require that universally quantified variables appear as arguments to uninterpreted symbols only
- The finite almost uninterpreted (FAU) fragment
  - Allow universally quantified variables to appear in arithmetic literals

Microsoft Ivy

- Programming language
  - Reactive – executes actions on response to input from environment
  - System state is a first order structure
  - Procedures generate first order formulæ that express how the state changes
- Hoare logic
  - Checks for preconditions before a procedure is executed and postconditions after it finishes

Microsoft Ivy

- Programming language
- Hoare logic
- Proof assistant
  - FAU fragment checker
    - Cycle detection for stratified function symbols post Skolemisation
    - Cycle detection for relevant vocabularies
  - Automated theorem prover
    - Z3 as a decision procedure for FAU

Microsoft Ivy

- Programming language
- Hoare logic
- Proof assistant
  - FAU fragment checker
  - Automated theorem prover
  - Inductive invariants
    - Check if they are satisfied in the initial state
    - Check that if you start from a state that satisfies the invariant and execute a procedure, the invariant is still satisfied

Microsoft Ivy

- Programming language
- Hoare logic
- Proof assistant
  - FAU fragment checker
  - Automated theorem prover
  - Inductive invariants
  - Manual proofs
    - Natural deduction built-in
    - Write our own tactics

Microsoft Ivy – Syntax

- `include` to include one file in another
- `isolate`s are independently verifiable
  - `with <isolate name>` to include invariants and properties from other isolates as axioms here
- `module`s group a bunch of declarations, procedures, and invariants
  - `instance` to create an object
- `object`s are modules with a single instance
  - `isolate`s also define `object`s

Microsoft Ivy – Syntax

- `include` to include one file in another
- `isolate`s are independently verifiable
- `module`s group a bunch of declarations, procedures, and invariants
- `object`s are modules with a single instance
- `type` to define new, uninterpreted sorts
  - `type this` inside an object to associate the contents as traits of the type

Microsoft Ivy – Syntax

- `include` to include one file in another
- `isolate`s are independently verifiable
- `module`s group a bunch of declarations, procedures, and invariants
- `object`s are modules with a single instance
- `type` to define new, uninterpreted sorts
- `action` defines procedures.
  - `export` to let the environment execute in arbitrary order for inductiveness check

Consensus Protocols

- The Byzantine Generals Problem (Lamport, 1982)
- State machine replication
- Byzantine fault-tolerant state machine replication
  - Classical result – Bound on the number of Byzantine nodes (Bracha and Toueg, 1985)
  - Classical result – no consensus in asynchronous networks (Fischer, Lynch, and Paterson, 1985)
- Paxos (Lamport, 1998)
- Practical Byzantine Fault Tolerance (PBFT) (Castro and Liskov, 1999)

Consensus Protocols – PBFT

- Three phases – propose*, prepare, commit
- Each round has a leader
- Leader gets request from client
- Leader broadcasts proposal
- Nodes vote on the proposal
- Nodes that get $2f + 1$ votes produce a prepare certificate and broadcast it
  - Guarantees uniqueness of proposal

*name changed from original paper to match SupraBFT

Consensus Protocols – PBFT

- Three phases – propose*, prepare, commit
- Each round has a leader
- Leader gets request from client
- …
- Nodes that get $2f + 1$ prepare certificates produce a commit certificate. In addition, they perform the necessary computation and send out a response to the client
  - Guarantees linearisability of requests and responses
- Extra mechanisms for timing out, synchronising after timing out

*name changed from original paper to match SupraBFT

Consensus Protocols – After PBFT

- Replace client request and response
  - Create a new block of transactions
  - Attach the block to the blockchain
- Tendermint (Buchman, 2016)
- HotStuff (Yin, Malkhi, Reiter, Gueta, and Abraham, 2019)
  - Introduced ‘chaining’ in the paper as a part of ‘Chained HotStuff’
  - Use prepare certificate for constructing next block while committing current
- Jolteon and Ditto (Gelashvili, Kokoris-Kogias, Sonnino, Spiegelman, and Xiang, 2022)

SupraBFT

- More chaining?
- Propose the next block, vote for the current block, commit the past block at the same time (kind of)
- When timing out, send the commit-candidate (locked) certificate
- When the next leader receives $2f + 1$ time-out messages, it picks the highest locked block and uses that in creating the next block
- When certificates are received out-of-order, create a broken chain of uncommitted blocks. When back-to-back blocks are available, try to extend the blockchain with the broken chain.

SupraBFT - Microsoft Ivy

- A model of the protocol in Ivy
- Cryptographic assumptions are modelled using traits on the types
- Byzantine behaviour is modelled using an exported action that can send arbitrary (but well-formed) messages
- For the proof, we also need to reason about properties that involve multiple node
  - A ‘global view’ of the protocol, that tracks various actions the nodes take on different blocks
- Network model – packets need to be sent for them to be received

Safety

- Main invariants
  - Round safety – If two blocks anywhere in the blockchain across different nodes have the same round, then they must be the same block
  - Safety – The block at every height in the blockchain is the same across nodes
- Invariants fail to be inductive due to starting in absurd states
  - Need supporting invariants strengthening existing ones
  - 300+ supporting invariants

Safety

- Main invariants
- Invariants fail to be inductive due to starting in absurd states
- Some interesting supporting invariants
  - All and only the things that happen in the local view happen in the global view (many invariants)
  - If a block has a quorum of nodes that voted on it, so does its parent
  - A block was added to the blockchain because of a valid locked block after it
  - Once a quorum of nodes agree upon something, they stay agreed upon on that forever
  - Every block in the blockchain is a descendent of every block above it

Further work

- This version of the algorithm is not live

Further work

- This version of the algorithm is not live ☹
  - Found through testing
  - Protocol is still in development at SupraOracles

Further work

- This version of the algorithm is not live ☹
  - Found through testing
  - Protocol is still in development at SupraOracles ☺
- Updating safety proof to correspond to protocol changes
- Proving liveness
  - New blocks keep getting added to the chain

Thank you