- Permutations
- States
- Protocols
- Reachable states
- Parameterized formulas, guarded commands, and protocols.
- Problem

Do the nodes, which run $\mathscr{P}$, satisfy some invariant requirement req?
Or, equivalently:
Does req hold in every reachable state of the nodes running $\mathscr{P}$?

工作流程

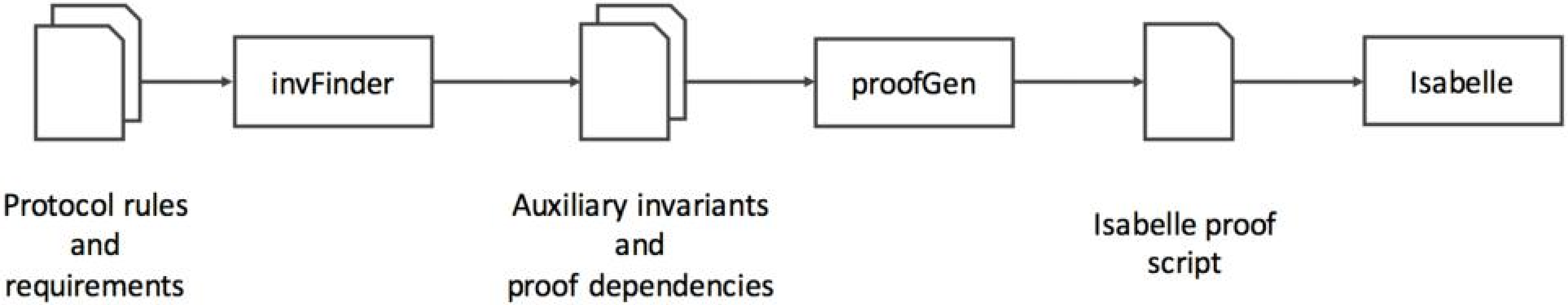

Fig. 1: The workflow of paraVerifier.

Inductive Invariant

A formula *inv* is an *inductive invariant* of protocol $\mathscr{P} = (I, R)$ if

— the initial state satisfies the formula: $\models I \to inv$, and

— transitions preserve the formula: for each $r \in R$, we have $\models inv \wedge \mathsf{guard}(r) \to \mathsf{WP}(\mathsf{action}(r), inv)$.

LEMMA 3.2. *Let $\mathscr{P} = (I, R)$ be a protocol. A formula inv is an inductive invariant of the protocol if and only if for all states s and s' and guarded commands $r \in R$ with $s \xrightarrow{r} s'$, $s \models inv$ implies $s' \models inv$.*

Protocol Requirement

PROPOSITION 3.3. *Assume given a protocol $\mathscr{P} = (I, R)$ and an (invariant) requirement req. $\mathscr{P}$ satisfies req if there exists an inductive invariant inv of $\mathscr{P}$ such that $\models inv \rightarrow req$.*

concretization strategy

- Concretizing parameter values for a requirement
  - Therefore, we simply insert 1,2,...,m as the node identities. We denote the concretizedrequirement by req^c.
- Concretizing parameter values for a guarded command
  - we cover every combination of nodes involved in the several roles of the requirement.

Generally, for a k-element list of node identities, we have to choose which ones of them are in $\{1, \ldots, m\}$, and set the remaining parameters to distinct values $> m$. This leads to $\sum_{j=0}^{\min\{k,m\}} \binom{k}{j} \frac{m!}{(m-j)!}$ possibilities.

Construct Concret Auxiliary Invariants

- casual relations

Recall that the invariant property we consider in this paper has the form $\neg \bigwedge_n f_n$, where each f_n is an atomic formula or predicate. Thus, $\mathsf{WP}(\mathsf{action}(r^{\mathrm{c}}), req^{\mathrm{c}})$ is a simple concrete formula of the same shape. If $\neg\mathsf{guard}(r^{\mathrm{c}})$ also has this shape, the formula $\neg\mathsf{guard}(r^{\mathrm{c}}) \vee \mathsf{WP}(\mathsf{action}(r^{\mathrm{c}}), req^{\mathrm{c}})$ can again be transformed into it.

We denote the obtained formula by $\neg \bigwedge_{n=1}^{k} f_n^{\mathrm{c}}$. Candidates for the concrete auxiliary invariant are formulas $\neg f$, where f is a conjunction of a few of the $f_1^{\mathrm{c}}, \ldots, f_k^{\mathrm{c}}$. Observe that f allows more states than the original formula and thus may not be an inductive invariant. Thus, we apply model checking to find a simple formula $\neg f$.

A semi-algorithm

- a semi-algorithm for finding proof dependencies as well as concretized candidates for strengthening

Genaralizing Concrete Invariants

- We generalize guarded commands in the context of a requirement and therefore have to add parameter constraints to describe the relations between the parameters of the requirement and those of the guarded command.

The second kind of generalization is to generalize a guarded command r^c in the context of a requirement $req(i_1, i_2, \ldots, i_m)$ and its concretization $req(1, 2, \ldots, m)$. Again, we can use the assumption that different parameters are assigned different node identities. The guarded command is a concretization of a general form $r(j_1, j_2, \ldots, j_\ell)$. Whenever we see that some parameter j has been instantiated by the node identity $n \in \{1, 2, \ldots, m\}$, we add the parameter constraint $j = i_n$. When, however, the parameter j has been instantiated with some value $>$ m, we add the parameter constraint $\bigwedge_{n=1}^{m} j \neq i_n$.