Probabilistic Guarded Commands Mechanized in HOL

Joe Hurd
joe.hurd@cl.cam.ac.uk
University of Cambridge

Joint work with Annabelle McIver (Macquarie University) and Carroll Morgan (University of New South Wales)
Contents

- **Introduction**
  - Formalizing Probabilistic Guarded Commands
  - \(wlp\) Verification Condition Generator
  - Example Verifications
  - Conclusion
Introduction

Probabilistic programs are useful for many applications:

- Symmetry breaking
  - Rabin’s mutual exclusion algorithm
- Eliminating pathological cases
  - Miller-Rabin primality test
- Algorithm complexity
  - Sorting nuts and bolts
- Defeating a powerful adversary
  - Mixed strategies in game theory
- Solving a problem in an extremely simple way
  - Finding minimal cuts
Introduction: pGCL

- pGCL stands for probabilistic Guarded Command Language.
- It’s Dijkstra’s GCL extended with probabilistic choice
  \[ c_1 \oplus p c_2 \]
- Like GCL, the semantics is based on weakest preconditions.
- Important: retains demonic choice
  \[ c_1 \sqcap c_2 \]
- Developed by Morgan et al. in the Programming Research Group, Oxford, 1994–
The HOL Theorem Prover

- Developed by Mike Gordon’s Hardware Verification Group in Cambridge, first release was HOL88.
- Latest release in mid-2002 called HOL4, developed jointly by Cambridge, Utah and ANU.
- Implements classical Higher-Order Logic with Hindley-Milner polymorphism.
- Sprung from the Edinburgh LCF project, so has a small logical kernel to ensure soundness.
- Links to external proof tools, either as oracles (e.g., SAT solvers) or by translating their proofs (e.g., Gandalf).
- Comes with a large library of theorems contributed by many users over the years, including theories of lists, real analysis, groups etc.
Contents

- Introduction
- **Formalizing Probabilistic Guarded Commands**
  - \(wlp\) Verification Condition Generator
  - Example Verifications
- Conclusion
pGCL Semantics

- Given a standard program $C$ and a postcondition $Q$, let $P$ be the weakest precondition that satisfies $\mathcal{L}[P]C[Q]$

- Precondition $P$ is weaker than $P'$ if $P' \Rightarrow P$.

- Such a $P$ will always exist and be unique, so think of $C$ as a function that transforms postconditions into weakest preconditions.

- pGCL generalizes this to probabilistic programs:
  - Conditions $\alpha \rightarrow \mathbb{B}$ become expectations $\alpha \rightarrow \text{posreal}$.
  - Expectation $P$ is weaker than $P'$ if $P' \subseteq P$.
  - Think of programs as expectation transformers.
Model pGCL commands with a HOL datatype:

\[
\text{command} \equiv \begin{cases} 
\text{Assert of } (\text{state} \rightarrow \text{posreal}) \times \text{command} \\
\text{Abort} \\
\text{Skip} \\
\text{Assign of string} \times (\text{state} \rightarrow \mathbb{Z}) \\
\text{Seq of } \text{command} \times \text{command} \\
\text{Demon of } \text{command} \times \text{command} \\
\text{Prob of } (\text{state} \rightarrow \text{posreal}) \times \text{command} \times \text{command} \\
\text{While of } (\text{state} \rightarrow \mathbb{B}) \times \text{command}
\end{cases}
\]

Note: the probability in \text{Prob} can depend on the state.
Derived Commands

Define the following derived commands as syntactic sugar:

\[ v := e \equiv \text{Assign } v e \]
\[ c_1 ; c_2 \equiv \text{Seq } c_1 c_2 \]
\[ c_1 \sqcap c_2 \equiv \text{Demon } c_1 c_2 \]
\[ c_1 p \oplus c_2 \equiv \text{Prob } (\lambda s. p) c_1 c_2 \]
\[ \text{Cond } b c_1 c_2 \equiv \text{Prob } (\lambda s. \text{if } b s \text{ then } 1 \text{ else } 0) c_1 c_2 \]
\[ v := \{e_1, \ldots, e_n\} \equiv v := e_1 \sqcap \cdots \sqcap v := e_n \]
\[ v := \langle e_1, \ldots, e_n \rangle \equiv v := e_1 1/n \oplus v := \langle e_2, \ldots, e_n \rangle \]
\[ p_1 \rightarrow c_1 | \cdots | p_n \rightarrow c_n \equiv \begin{cases} \text{Abort} & \text{if none of the } p_i \text{ hold on the current state} \\ \prod_{i \in I} c_i & \text{where } I = \{i \mid 1 \leq i \leq n \land p_i \text{ holds} \} \end{cases} \]

In addition, we write \( v := n + 1 \) instead of \( \text{“} v \text{”} := \lambda s. s \text{ “} n \text{”} + 1. \)
Weakest Preconditions

Define weakest preconditions ($wp$) directly on commands:

$$
\vdash (wp \text{ Assert } p c) = wp c
\land (wp \text{ Abort } = \lambda r. \text{ Zero})
\land (wp \text{ Skip } = \lambda r. r)
\land (wp \text{ Assign } v e) = \lambda r, s. r (\lambda w. \text{ if } w = v \text{ then } e s \text{ else } s w))
\land (wp \text{ Seq } c_1 c_2) = \lambda r. wp c_1 (wp c_2 r))
\land (wp \text{ Demon } c_1 c_2) = \lambda r. \text{ Min } (wp c_1 r) (wp c_2 r))
\land (wp \text{ Prob } p c_1 c_2) =
\begin{array}{l}
\lambda r, s. \text{ let } x \leftarrow [p s]_{\leq 1} \text{ in } x(wp c_1 r s) + (1 - x)(wp c_2 r s))
\land (wp \text{ While } b c) =
\lambda r. \text{ expect}_lfp (\lambda e, s. \text{ if } b s \text{ then } wp c e s \text{ else } r s))
\end{array}
$$
Weakest Preconditions: Example

- The goal is to end up with variables $i$ and $j$ containing the same value:

\[
post \equiv \text{if } i = j \text{ then } 1 \text{ else } 0.
\]

- First program:

\[
\text{pd} \equiv i := \{0, 1\}; \ j := \{0, 1\}
\]
\[
\vdash \text{wp pd post} = \text{Zero}
\]

- Second program:

\[
\text{dp} \equiv j := \{0, 1\}; \ i := \{0, 1\}
\]
\[
\vdash \text{wp dp post} = \lambda s. \ 1/2.
\]
Contents

- Introduction
- Formalizing Probabilistic Guarded Commands
- \texttt{wlp Verification Condition Generator}
- Example Verifications
- Conclusion
Weakest Liberal Preconditions

Weakest liberal conditions ($\text{wlp}$) model partial correctness.

\[
\vdash (\text{wlp} \ (\text{Assert} \ p \ c) = \text{wlp} \ c) \\
\land (\text{wlp} \ \text{Abort} = \lambda r. \ \text{Magic}) \\
\land (\text{wlp} \ \text{Skip} = \lambda r. \ r) \\
\land (\text{wlp} \ (\text{Assign} \ v \ e) = \lambda r, s. \ r \ (\lambda w. \ \text{if} \ w = v \ \text{then} \ e \ s \ \text{else} \ s \ w)) \\
\land (\text{wlp} \ (\text{Seq} \ c_1 \ c_2) = \lambda r. \ \text{wlp} \ c_1 \ (\text{wlp} \ c_2 \ r)) \\
\land (\text{wlp} \ (\text{Demon} \ c_1 \ c_2) = \lambda r. \ \text{Min} \ (\text{wlp} \ c_1 \ r) \ (\text{wlp} \ c_2 \ r)) \\
\land (\text{wlp} \ (\text{Prob} \ p \ c_1 \ c_2) = \\
\quad \lambda r, s. \ \text{let} \ x \leftarrow [p \ s]_{\leq 1} \ \text{in} \ x(\text{wlp} \ c_1 \ r \ s) + (1 - x)(\text{wlp} \ c_2 \ r \ s)) \\
\land (\text{wlp} \ (\text{While} \ b \ c) = \\
\quad \lambda r. \ \text{expect}_\text{gfp} \ (\lambda e, s. \ \text{if} \ b \ s \ \text{then} \ \text{wlp} \ c \ e \ s \ \text{else} \ r \ s))
\]
Weakest Liberal Preconditions: Example

- We illustrate the difference between $wp$ and $wlp$ on the simplest infinite loop:

$$\text{loop } \equiv \text{While } (\lambda s. \top) \text{ Skip}$$

- For any postcondition $post$, we have

$$\vdash wp \text{ loop } post = \text{Zero} \land wlp \text{ loop } post = \text{Magic}$$

- These correspond to the Hoare triples

$$\langle \bot \rangle \text{ loop } \langle post \rangle \quad \langle \top \rangle \text{ loop } \{ post \}$$

as we would expect from an infinite loop.
Calculating \( w \vdash_p \) Lower Bounds

- Suppose we have a pGCL command \( c \) and a postcondition \( q \).
- We wish to derive a lower bound on the weakest liberal precondition.
- Can think of this as the first-order query \( P \sqsubseteq w \vdash_p c \ q \).
- **Idea:** use a Prolog interpreter to solve for the variable \( P \).
Calculating \( wlp \): Rules

Example Rules:

- Magic \( \sqsubseteq wlp \) Abort \( Q \)
- \( Q \sqsubseteq wlp \) Skip \( Q \)
- \( R \sqsubseteq wlp \) \( C_2 \) \( Q \) \& \( P \sqsubseteq wlp \) \( C_1 \) \( R \) \( \Rightarrow \)
  \( P \sqsubseteq wlp \) (Seq \( C_1 \) \( C_2 \)) \( Q \)
- \( P_1 \sqsubseteq wlp \) \( C_1 \) \( Q \) \& \( P_2 \sqsubseteq wlp \) \( C_2 \) \( Q \) \( \Rightarrow \)
  Min \( P_1 \) \( P_2 \) \( \sqsubseteq wlp \) (Demon \( C_1 \) \( C_2 \)) \( Q \)

Note: the Prolog interpreter automatically calculates the ‘middle condition’ in a Seq command.
Calculating $wlp$: While Loops

- We use the following theorem about While loops:

  $\vdash \forall P, Q, b, c. \ P \sqsubseteq \text{if } b \ (wlp \ c \ P) \ Q \Rightarrow P \sqsubseteq wlp\ \text{(While } b \ c) \ Q$

- Cannot use in this form, because of the repeated occurrence of $P$ in the premise.

- Instead, provide a rule that requires an assertion:
  
  - $R \sqsubseteq wlp\ C \ P \land P \sqsubseteq \text{if } B \ R \ Q \Rightarrow P \sqsubseteq wlp\ (\text{Assert } P \ (\text{While } B \ C')) \ Q$

- The second premise generates a verification condition as an extra subgoal.

- It is left to the user to provide a useful loop invariant in the Assert around the while loop.
Contents

- Introduction
- Formalizing Probabilistic Guarded Commands
- \textit{wlp} Verification Condition Generator
- Example Verifications
- Conclusion
Example: Monty Hall

contestant \textbf{switch} \equiv

\begin{align*}
  pc &:= \{1, 2, 3\} ; \\
  cc &:= \langle 1, 2, 3 \rangle ; \\
  &pc \neq 1 \land cc \neq 1 \rightarrow ac := 1 \\
  | &pc \neq 2 \land cc \neq 2 \rightarrow ac := 2 \\
  | &pc \neq 3 \land cc \neq 3 \rightarrow ac := 3 ;
\end{align*}

if \neg \textbf{switch} then Skip else

cc := (if cc \neq 1 \land ac \neq 1 then 1 \\
       \quad \text{else if } cc \neq 2 \land ac \neq 2 \text{ then } 2 \text{ else } 3)

The postcondition is simply the desired goal of the contestant, i.e.,

\begin{align*}
  \textbf{win} &\equiv \text{if } cc = pc \text{ then } 1 \text{ else } 0 .
\end{align*}
Example: Monty Hall

- Verification proceeds by:
  1. Rewriting away all the syntactic sugar.
  2. Expanding the definition of $wp$.
  3. Carrying out the numerical calculations.

- After 22 seconds and 250536 primitive inferences in the logical kernel:

\[
\vdash wp\ (\text{contestant } switch) \text{ win} = \lambda s. \text{ if } switch \text{ then } 2/3 \text{ else } 1/3
\]

- In other words, by switching the contestant is twice as likely to win the prize.

- Not trivial to do by hand, because the intermediate expectations get rather large.
Example: Rabin Mutual Exclusion

- Suppose $N$ processors are executing concurrently, and from time to time some of them need to enter a critical section of code.

- The mutual exclusion algorithm of Rabin (1982, 1992) works by electing a leader who is permitted to enter the critical section:
  1. Each of the waiting processors repeatedly tosses a fair coin until a head is shown.
  2. The processor that required the largest number of tosses wins the election.
  3. If there is a tie, then have another election.

- Could implement the coin tossing using
  
  $$n := 0 \; ; \; b := 0 \; ; \; \text{While} \; (b = 0) \; (n := n + 1 \; ; \; b := \langle 0, 1 \rangle)$$
Example: Rabin Mutual Exclusion

For our verification, we do not model $i$ processors concurrently executing the above voting scheme, but rather the following data refinement of that system:

1. Initialize $i$ with the number of processors waiting to enter the critical section who have just picked a number.
2. Initialize $n$ with 1, the lowest number not yet considered.
3. If $i = 1$ then we have a unique winner: return SUCCESS.
4. If $i = 0$ then the election has failed: return FAILURE.
5. Reduce $i$ by eliminating all the processors who picked the lowest number $n$ (since certainly none of them won the election).
6. Increment $n$ by 1, and jump to Step 3.
Example: Rabin Mutual Exclusion

The following pGCL program implements this data refinement:

\[
\text{rabin} \equiv \text{While } (1 < i) ( \\
\hspace{1cm} n := i ; \\
\hspace{1cm} \text{While } (0 < n) \\
\hspace{2cm} (d := \langle 0, 1 \rangle ; i := i - d ; n := n - 1) \\
\)
\]

The desired postcondition representing a unique winner of the election is

\[
\text{post} \equiv \text{if } i = 1 \text{ then } 1 \text{ else } 0
\]
Example: Rabin Mutual Exclusion

- The precondition that we aim to show is

\[ \text{pre} \equiv \text{if } i = 1 \text{ then 1 else if } 1 < i \text{ then } 2/3 \text{ else 0} \]

“For any positive number of processors wanting to enter the critical section, the probability that the voting scheme will produce a unique winner is 2/3, except for the trivial case of one processor when it will always succeed.”

- **Surprising:** The probability of success is independent of the number of processors.

- We formally verify the following statement of partial correctness:

\[ \text{pre} \sqsubseteq \text{wlp rabin post} \]
Example: Rabin Mutual Exclusion

- Need to annotate the While loops with invariants.
- The invariant for the outer loop is simply \textit{pre}.
- For the inner loop we used

\[
\text{if } 0 \leq n \leq i \text{ then } (2/3) \times \text{invar1 } i \ n + \text{invar2 } i \ n \text{ else } 0
\]

where

\[
\text{invar1 } i \ n \equiv 1 - (\text{if } i = n \text{ then } (n + 1)/2^n \text{ else if } i = n + 1 \text{ then } 1/2^n \text{ else } 0)
\]

\[
\text{invar2 } i \ n \equiv \text{if } i = n \text{ then } n/2^n \text{ else if } i = n + 1 \text{ then } 1/2^n \text{ else } 0
\]

- Coming up with these was the hardest part of the verification.
Example: Rabin Mutual Exclusion

The verification proceeded as follows:

1. Create the annotated program annotated_rabin.
2. Prove \( wlp \ rabin = wlp \ \text{annotated}_\ rabin \)
3. Use this to reduce the goal to

\[
\text{pre} \subseteq wlp \ \text{annotated}_\ rabin \ \text{post}
\]

4. This is in the correct form to apply the VC generator.
5. Finish off the VCs with 58 lines of HOL-4 proof script.

\[
\text{|- Leq (}\forall s. \ \text{if } s"i" = 1 \ \text{then } 1 \\
\quad \ \text{else if } 1 < s"i" \ \text{then } 2/3 \ \text{else } 0) \\
\quad (wlp \ \text{rabin (}\forall s. \ \text{if } s"i" = 1 \ \text{then } 1 \ \text{else } 0))
\]
Contents

- Introduction
- Formalizing Probabilistic Guarded Commands
- $\omega$lp Verification Condition Generator
- Example Verifications
- Conclusion
Conclusion

- Formalized the theory of pGCL in higher-order logic.
  - Definitional theory, so high assurance of consistency.
  - Created the first direct proof that \( wp \) semantics always give healthy transformers.
- Created an automatic tool for deriving sufficient conditions for partial correctness.
  - Useful product of mechanizing a program semantics.
  - Used in a verification of the probabilistic voting scheme in Rabin’s mutual exclusion algorithm.
- HOL-4 well suited to this task.
  - Hard VCs can be passed to the user as subgoals.
  - LCF kernel enforces soundness, even though the VC generator tactic is a highly complex program.
Related Work

- Formal methods for probabilistic programs:
  - Probabilistic invariants for probabilistic machines, Hoang et. al., 2003.
  - Christine Paulin’s work in Coq, 2002.
  - Prism model checker, Kwiatkowska et. al., 2000–

- Mechanized program semantics:
  - Mechanizing program logics in higher order logic, Gordon, 1989.