Probabilistic Guarded Commands
Mechanized in HOL

Joe Hurd
joe.hurd@comlab.ox.ac.uk

Oxford University

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: Rabin’s Mutual Exclusion Algorithm
  - Conclusion
Introduction: pGCL

- pGCL stands for probabilistic Guarded Command Language.
- It’s Dijkstra’s GCL extended with probabilistic choice

\[ c_1 \ p\oplus \ c_2 \]

- Like GCL, the semantics is based on weakest preconditions.
- **Important:** retains demonic choice

\[ c_1 \ \Box \ c_2 \]

- 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 called HOL4, developed jointly by Cambridge, Utah and ANU.
- Implements classical Higher-Order Logic: essentially first-order logic with quantification over functions.
- Sprung from the Edinburgh LCF project, so has a small logical kernel to ensure soundness.
Contents

- Introduction
- **Formalizing Probabilistic Guarded Commands**
  - $\text{wlp}$ Verification Condition Generator
  - Example: Rabin’s Mutual Exclusion Algorithm
- Conclusion
pGCL Semantics

- Given a standard GCL program $C$ and a postcondition $Q$, let $P$ be the weakest precondition that satisfies $[P]C[Q]$

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

- 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 $\alpha \rightarrow [0, +\infty]$.
  - Expectation $P$ is weaker than $P'$ if $P' \sqsubseteq P$.
  - Think of programs as expectation transformers.
Expectations

- Expectations are reward functions, from states to expected rewards.
- Modelled in HOL as functions $\alpha \rightarrow [0, +\infty]$.
- Define the following operations on expectations:
  - $\text{Min } e_1 e_2 \equiv \lambda s. \min (e_1 s) (e_2 s)$
  - $e_1 \sqsubseteq e_2 \equiv \forall s. e_1 s \leq e_2 s$
  - $\text{Cond } b e_1 e_2 \equiv \lambda s. \text{if } b s \text{ then } e_1 s \text{ else } e_2 s$
  - $\text{Lin } p e_1 e_2 \equiv \lambda s. [p s]^{\leq 1} \times e_1 s + (1 - [p s]^{\leq 1}) \times e_2 s$
States

- Fix states to be mappings from variable names to integers:

  \[ \text{state} \equiv \text{string} \rightarrow \mathbb{Z} \]

- For convenience, define a state update function:

  \[ \text{assign } v \ f \ s \equiv \lambda w. \text{ if } v = w \text{ then } f s \text{ else } s w \]
Model pGCL commands with a HOL datatype:

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

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

Define the following derived commands as syntactic sugar:

\[
\begin{align*}
v & := f \quad \equiv \quad \text{Assign } v \leftarrow f \\
c_1 ; c_2 & \quad \equiv \quad \text{Seq } c_1 \quad c_2 \\
c_1 \triangleright c_2 & \quad \equiv \quad \text{Demon } c_1 \quad c_2 \\
c_1 p\ominus c_2 & \quad \equiv \quad \text{Prob } (\lambda s. p) \quad c_1 \quad c_2 \\
\text{Cond } b \quad c_1 \quad c_2 & \quad \equiv \quad \text{Prob } (\lambda s. \text{if } b \quad s \quad \text{then } 1 \quad \text{else } 0) \quad c_1 \quad c_2 \\
v & := \{e_1, \ldots, e_n\} \quad \equiv \quad v := e_1 \quad \underline{\cap} \quad \cdots \quad \underline{\cap} \quad v := e_n \\
v & := \langle e_1, \ldots, e_n\rangle \quad \equiv \quad v := e_1 \quad \underline{1/n}\ominus \quad v := \langle e_2, \ldots, e_n\rangle \\
b_1 \rightarrow c_1 \mid \cdots \mid b_n \rightarrow c_n & \quad \equiv \quad \\
\begin{cases}
\text{Abort} & \text{if none of the } b_i \text{ hold on the current state} \\
\Pi_{i \in I} c_i & \text{where } I = \{i \mid 1 \leq i \leq n \land b_i \text{ holds}\}
\end{cases}
\end{align*}
\]
Weakest Preconditions

Define weakest preconditions ($\text{wp}$) directly on commands:

- ($\text{wp } \text{Abort} = \lambda e. \text{Zero})$
- ($\text{wp } \text{Skip} = \lambda e. e$)
- ($\text{wp } (\text{Assign } v f) = \lambda e, s. e \oplus (\text{assign } v f s)$)
- ($\text{wp } (\text{Seq } c_1 c_2) = \lambda e. \text{wp } c_1 (\text{wp } c_2 e)$)
- ($\text{wp } (\text{Demon } c_1 c_2) = \lambda e. \text{Min } (\text{wp } c_1 e) (\text{wp } c_2 e)$)
- ($\text{wp } (\text{Prob } p c_1 c_2) = \lambda e. \text{Lin } p (\text{wp } c_1 e) (\text{wp } c_2 e)$)
- ($\text{wp } (\text{While } b c) = \lambda e. \text{expect}_{\text{lfp}} (\lambda e'. \text{Cond } b (\text{wp } c e') e)$)
Weakest Preconditions: Example

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

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

- First program:

\[
\begin{align*}
\text{pd} & \equiv i := \langle 0, 1 \rangle ; j := \{0, 1\} \\
\vdash \text{wp pd post} & = \text{Zero}
\end{align*}
\]

- Second program:

\[
\begin{align*}
\text{dp} & \equiv j := \{0, 1\} ; i := \langle 0, 1 \rangle \\
\vdash \text{wp dp post} & = \lambda s. 1/2.
\end{align*}
\]
Example: Monty Hall

contestant \textit{switch} \equiv

\begin{align*}
pc & := \{1, 2, 3\} ; \\
cc & := \langle 1, 2, 3 \rangle ; \\
& \quad pc \neq 1 \land cc \neq 1 \rightarrow ac := 1 \\
& \quad | \quad pc \neq 2 \land cc \neq 2 \rightarrow ac := 2 \\
& \quad | \quad pc \neq 3 \land cc \neq 3 \rightarrow ac := 3 ; \\
\text{if } \neg \textit{switch} \text{ then Skip else} \\
cc & := (\text{if } cc \neq 1 \land ac \neq 1 \text{ then } 1 \\
& \quad \text{else if } cc \neq 2 \land ac \neq 2 \text{ then } 2 \text{ else } 3) \\
\end{align*}

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

\[ \text{win } \equiv \text{ if } cc = pc \text{ then } 1 \text{ else } 0. \]
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:

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

- 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.
Contents

- Introduction
- Formalizing Probabilistic Guarded Commands
- \( \text{wlp} \) Verification Condition Generator
- Example: Rabin’s Mutual Exclusion Algorithm
- Conclusion
Weakest liberal preconditions model partial correctness.

\[ \vdash (\text{wlp \ Abort} = \lambda e. \text{Infty}) \]
\[ \land (\text{wlp \ Skip} = \lambda e. e) \]
\[ \land (\text{wlp \ (Assign \ v \ f)} = \lambda e, s. e (\text{assign \ v \ f \ s)}) \]
\[ \land (\text{wlp \ (Seq \ c_1 \ c_2)} = \lambda e. \text{wlp \ c_1 \ (wlp \ c_2 \ e)}) \]
\[ \land (\text{wlp \ (Demon \ c_1 \ c_2)} = \lambda e. \text{Min \ (wlp \ c_1 \ e) \ (wlp \ c_2 \ e)}) \]
\[ \land (\text{wlp \ (Prob \ p \ c_1 \ c_2)} = \lambda e. \text{Lin \ p \ (wlp \ c_1 \ e) \ (wlp \ c_2 \ e)}) \]
\[ \land (\text{wlp \ (While \ b \ c)} = \lambda e. \text{expect_gfp} (\lambda e'. \text{Cond \ b \ (wlp \ c \ e')} \ e)) \]
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

\[
\vdash \ wp \ \text{loop} \ post = \text{Zero} \land \ wlp \ \text{loop} \ post = \text{Infty}
\]

- These correspond to the Hoare triples

\[
[\bot] \text{loop} \ [post] \quad \{\top\} \text{loop} \ \{post\}
\]

as we would expect from an infinite loop.
Calculating $wlp$ 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.
  - In general, programs are shown to have desirable properties by proving *lower bounds*.
  - Example: $\vdash (\lambda s. 0.95) \sqsubseteq wp \text{Prog} \text{ (if ok then 1 else 0)}$
- Can think of this as the query $P \sqsubseteq wlp \ c \ q$.
- **Idea:** use a Prolog interpreter to solve for the variable $P$. 
Calculating \( wlp \): Rules

Simple rules:

- \( \text{Infty} \subseteq wlp \text{ Abort } Q \)
- \( Q \subseteq wlp \text{ Skip } Q \)
- \( R \subseteq wlp C_2 Q \land P \subseteq wlp C_1 R \)
  \[ \Rightarrow \]
  \[ P \subseteq wlp (\text{Seq } C_1 C_2) Q \]

**Note:** the Prolog interpreter automatically calculates the ‘middle condition’ in a \( \text{Seq} \) command.
**Calculating \( wlp \): While Loops**

- Define an assertion command: Assert \( p \ c \ \equiv \ c \).
- Provide a while rule that requires an assertion:
  
  \[
  R \subseteq wlp \ C \ P \land P \subseteq wlp\_\text{cond} \ b \ R \ Q \\
  \Rightarrow \\
  P \subseteq wlp (\text{Assert} \ P (\text{While} \ b \ c)) \ Q
  \]

- 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
- \texttt{wlp} Verification Condition Generator
  - Example: Rabin’s Mutual Exclusion Algorithm
- Conclusion
Rabin’s Mutual Exclusion Algorithm

- 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 ; While (b = 0) (n := n + 1 ; b := ⟨0, 1⟩)
  ```
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.
Rabin’s Mutual Exclusion Algorithm

The following pGCL program implements this data refinement:

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

The desired postcondition representing a unique winner of the election is

\[
\text{post } \equiv \text{ if } i = 1 \text{ then } 1 \text{ else } 0
\]
Rabin’s Mutual Exclusion Algorithm

- The precondition that we aim to show is

  \[ pre \equiv \begin{cases} 
  1 & \text{if } i = 1 \\
  2/3 & \text{if } 1 < i \\
  0 & \text{else} 
  \end{cases} \]

  “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:

  \[ pre \sqsubseteq wlp \text{ rabin } post \]
Rabin’s Mutual Exclusion Algorithm

- Need to annotate the While loops with invariants.
- The invariant for the outer loop is simply \( \text{pre} \).
- 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.
Rabin’s Mutual Exclusion Algorithm

The verification proceeded as follows:

1. Create the annotated program \texttt{annotated\_rabin}.
2. Prove \texttt{rabin = annotated\_rabin}
3. Use this to reduce the goal to

   \[\text{\texttt{pre} \sqsupseteq \text{wlp annotated\_rabin post}}\]

4. This is now in the correct form to apply the VC generator.

5. Finish off the VCs with 58 lines of HOL-4 proof script.

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

- Introduction
- Formalizing Probabilistic Guarded Commands
- $\omega_l p$ Verification Condition Generator
- Example: Rabin’s Mutual Exclusion Algorithm

Conclusion
Conclusion

- Formulated the theory of pGCL in higher-order logic.
  - Definitional theory, so high assurance of consistency.

- Created an automatic tool for deriving sufficient conditions for partial correctness.
  - Useful product of mechanizing a program semantics.

- HOL-4 well suited to this task.
  - Hard VCs can be passed to the user as subgoals.
Related Work

- Formal methods for probabilistic programs:
  - 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.