Proofs and refutations

for probabilistic systems

Annabelle McIver
Carroll Morgan
Carlos Gonzalia
In a nutshell...

Probabilistic programs are tricky. Even small ones can be hard to grasp intuitively, especially if nondeterminism is also present.

Mimicking the proving and refuting — so well understood conventionally — is not easy for demonic and probabilistic programs.

Why isn’t it easy? How do we get a grip on it? How do we automate it?
Probabilistic programs are tricky. Even small ones can be hard to grasp, especially if nondeterminism is present.

Mimicking the proving and refuting — so well understood conventionally — is not easy for demonic and probabilistic programs.

Why isn’t it easy? How do we get a grip on it? How do we automate it?
Main points: a trilogy

• Why are probabilistic proofs and refutations not straightforward? What’s hard.

• What’s a good model for probabilistic computation? What’s easy.

• Automating proofs and refutations in that model. What’s sneaky.
pGCL syntax

**skip**

**abort**

\[ x := E \]

\[ \text{prog}_1 \bigoplus \text{prog}_2 \]

\[ \text{prog}_1 \sqcap \text{prog}_2 \]

\[ \text{prog}_1; \text{prog}_2 \]

Prelude.

Do nothing

Do absolutely anything

Assignment

Probabilistic choice

Demonic choice

Sequential composition
pGCL semantics

skip
abort

\[ x := E \]

\[ \text{prog}_1 \oplus \text{prog}_2 \]

\[ \text{prog}_1 \sqcap \text{prog}_2 \]

\[ \text{prog}_1 ; \text{prog}_2 \]

Demonic choice, completely unpredictable.
**pGCL semantics**

- **skip**
- **abort**
- $x := E$
- $prog_1 \oplus p \oplus prog_2$
- $prog_1 \sqcap prog_2$
- $prog_1; prog_2$

Probabilistic choice, where $p$ can be an expression.
Counter-examples

\[ s := X \sqcap (s := Y^{1/2} \oplus Z) \]
Counter-examples

\[ s := X \sqcap \left( s := Y^{1/2} \oplus Z \right) \]
Counter-examples

\[ A \quad s := X \sqcap (s := Y \frac{1}{2} \oplus Z) \]
\[ B \quad (s := X \sqcap Y) \frac{1}{2} \oplus (s := X \sqcap Z) \]

How do \( A \) and \( B \) compare?

Does \( B \) refine \( A \)? \( A \sqsubseteq B \)

Does \( A \) refine \( B \)? \( A \sqsupseteq B \)

Are they equal? \( A = B \)
Counter-examples

What’s hard.

\[
A \quad s := X \cap (s := Y_{1/2} \oplus Z)
\]

\[
B \quad (s := X \cap Y)_{1/2} \oplus (s := X \cap Z)
\]

How do \( A \) and \( B \) compare?

Does \( B \) refine \( A \)? \( A \sqsubseteq B \)

Does \( A \) refine \( B \)? \( A \sqsupseteq B \)

Are they equal? \( A = B \)
Counter-examples

\[ s := X \sqcap (s := Y_{1/2} \oplus Z) \]

\[ (s := X \sqcap Y)_{1/2} \oplus (s := X \sqcap Z) \]

How do A and B compare?

Does B refine A? \( A \sqsubseteq B \) ?

Does A refine B? \( A \sqsubseteq B \) ?

Are they equal? \( A \neq B \)
Let the state space $S$ be $\{X, Y, Z\}$; then a (discrete) distribution over $S$ is a one-summing triple of probabilities $(p_x, p_y, p_z)$.

Any point on the interior of an equilateral triangle can be uniquely defined as a linear combination of its vertices. We can use points within such a triangle to represent distributions—the outcomes of our programs—over a three-element state space.

* Kozen then Jones, Plotkin then He, McIver, Seidel, Morgan and independently Segala, Lynch.
Model* for pGCL

Let the state space $S$ be \{X, Y, Z\}; then a (discrete) distribution over $S$ is a one-summing triple of probabilities $(p_x, p_y, p_z)$.

Any point on the interior of an equilateral triangle is a unique linear combination of its vertices; so we can use points within such a triangle to represent distributions — the outcomes of our programs — over a three-element state space.

* Kozen then Jones, Plotkin then He, McIver, Seidel, Morgan and independently Segala, Lynch.
Model for pGCL

What’s easy.
Model for pGCL

What’s easy.

Discrete probability space over three values.
Model for pGCL

What’s easy.
Model for $pGCL$

What’s easy.
Model for pGCL

What’s easy.

(1,0,0)

(1,1/3,1/3,1/3)

(0,1/2,1/2)

Z

X

Y

10
Model for pGCL

What’s easy.

Example

Poor man’s fairness: Allow every distribution in which any one value is never more than twice as probable as any other value.
Model for pGCL

What’s easy.

Example

Poor man’s fairness: Allow every distribution in which any one value is never more than twice as probable as any other value.
Model for \( pGCL \)

Example

Poor man’s fairness: Allow every distribution in which any one value is never more than twice as probable as any other value.

\[ (1/4,1/4,1/2) \]

\[ (2/5,2/5,1/5) \]
Model for pGCL

What’s easy.

Example

Poor man’s fairness: Allow every distribution in which any one value is never more than twice as probable as any other value.
Example

Poor man’s fairness: Allow every distribution in which any one value is never more than twice as probable as any other value.
Example

Poor man’s fairness: Allow every distribution in which any one value is never more than twice as probable as any other value.
Model for $pGCL$

Demonic choice creates solid lines and regions.
Model for $pGCL$

Demonic choice creates solid lines and regions.

$s := X \cap (s := Y^{1/2} \oplus Z)$
Model for $pGCL$

What’s easy.

Demonic choice creates solid lines and regions.

$s := X \cap (s := Y_{1/2} \oplus Z)$
Model for $pGCL$

What’s easy.

Demonic choice creates solid lines and regions.

$$s := X \cap (s := Y \frac{1}{2} \oplus Z)$$
Model for pGCL

What’s easy.

Demonic choice creates solid lines and regions.
Demonic choice creates solid lines and regions.

What’s easy.

Model for pGCL

\[(s := X \cap Y) \frac{1}{2} \oplus (s := X \cap Z)\]
Demonic choice creates solid lines and regions.
Model for \( pGCL \)

Demonic choice creates solid lines and regions.
Demonic choice creates solid lines and regions.
Model for $pGCL$

What’s easy.

Demonic choice creates solid lines and regions.
Model for $pGCL$

Refinement is easy.
Proofs and refutations

What’s sneaky.

\[ p_x \leq 2p_y, 2p_z \]
\[ p_y \leq 2p_z, 2p_x \]
\[ p_z \leq 2p_x, 2p_y \]
Proofs and refutations

What’s sneaky.

$$p_x \leq 2p_y, 2p_z$$
$$p_y \leq 2p_z, 2p_x$$
$$p_z \leq 2p_x, 2p_y$$

$$p_x \geq 0.3$$
$$p_y \geq 0.3$$
$$p_z \geq 0.3$$

Poor man’s implementation
Proofs and refutations

What’s sneaky.

\[ \begin{align*}
    p_x & \leq 2p_y, 2p_z \\
    p_y & \leq 2p_z, 2p_x \\
    p_z & \leq 2p_x, 2p_y
\end{align*} \]

\[ \begin{align*}
    p_x & \geq 0.3 \\
    p_y & \geq 0.3 \\
    p_z & \geq 0.3
\end{align*} \]

All shapes produced by finite-state \( pGCL \) programs, even loops, are convex and finitely vertexed.

Poor man’s implementation
Proofs and refutations

What's sneaky.

For non-looping programs, those vertices can be calculated automatically from the source code. We use *The Jackal*.
Proofs and refutations

What’s sneaky.

To show refinement, in this case $C \subseteq D$, simply exhibit each of $D$'s three vertices as a linear combination, an \textit{interpolation}, of $C$'s six.

\[
p_x \leq 2p_y, 2p_z \\
p_y \leq 2p_z, 2p_x \\
p_z \leq 2p_x, 2p_y
\]

\[
p_x \geq 0.3 \\
p_y \geq 0.3 \\
p_z \geq 0.3
\]
To show refinement, in this case $C \subseteq D$, simply exhibit each of $D$'s three vertices as a linear combination, an *interpolation*, of $C$'s six.
Proofs and refutations

What’s sneaky.

Finding the interpolation can be done by an SMT solver. We use Yices.
Proofs and refutations

What’s sneaky.

Finding the interpolation can be done by an SMT solver. We use Yices.

Poor man’s implementation
Proofs and refutations

What’s sneaky.

(0.3, 0.3, 0.4) = (0.5, 0.5) \times \begin{pmatrix} 1/5 & 2/5 & 2/5 \\ 2/5 & 1/5 & 2/5 \end{pmatrix}
Proofs and refutations

What's sneaky.

\[(0.3, 0.3, 0.4) = (0.5, 0.5) \times \begin{pmatrix} 1/5 & 2/5 & 2/5 \\ 2/5 & 1/5 & 2/5 \end{pmatrix} \]

C

- \((1/5, 2/5, 2/5)\)
- \((2/5, 1/5, 2/5)\)

D

\((0.3, 0.3, 0.4) \Delta\)
Proofs and refutations

What’s sneaky.

\[(0.3, 0.3, 0.4) = (0.5, 0.5) \times \begin{pmatrix} 1/5 & 2/5 & 2/5 \\ 2/5 & 1/5 & 2/5 \end{pmatrix} \]
The interpolation \((0.5, 0.5)\) is the \textit{affirmation certificate} and it can be verified \textit{independently} — hard to find, but easy to check.
But what could a *refutation certificate* be? Failure of some SMT solver is, in itself, not enough: another one might have done better.
Proofs and refutations

What’s sneaky.

No interpolation of blue vertices will do.

(0.2, 0.2, 0.6) △
The *Separating Hyperplane Lemma* says that any point outside a closed convex shape can be separated from it by a (hyper-) plane.

No interpolation of blue vertices will do.

The *Separating Hyperplane Lemma* says that any point outside a closed convex shape can be separated from it by a (hyper-) plane.
Proofs and refutations

What’s sneaky.

Here the “plane” is $z = 0.55$, and the refutation certificate is its coefficients and constant term, thus $(0,0,1 \mid 0.55)$. 

No interpolation of blue vertices will do.

$(0.2, 0.2, 0.6) \Delta$
Proofs and refutations

An SMT solver finds that certificate; but it is *independently verifiable* that \((0.2, 0.2, 0.6)\) is above, and all blue points \((?, ?, \leq 0.5)\) are below.
Source-level refutation

\[ A \quad s := X \sqcap (s := Y \ 1/2 \oplus Z) \]
\[ B \quad (s := X \sqcap Y) \ 1/2 \oplus (s := X \sqcap Z) \]

The automatic refutation procedure (Jackal+Yices) for the claimed refinement \( A \sqsubseteq B \) produces the line shown (or similar), which is \( 7x + 3y + 12z = 6 \). The refutation certificate is tuple \((7, 3, 12 | 6)\).
Source-level refutation

\[ A \quad s := X \quad \cap \quad (s := Y \quad \frac{1}{2} \oplus Z) \]

\[ B \quad (s := X \quad \cap Y) \quad \frac{1}{2} \oplus (s := X \quad \cap Z) \]

The automatic refutation procedure \((Jackal+Yices)\) for the claimed refinement \(A \sqsubseteq B\) produces the line shown (or similar), which is \(7x + 3y + 12z = 6\). The refutation certificate is tuple \((7, 3, 12 \mid 6)\).
Source-level refutation

\[ A \quad s := X \quad \cap \quad (s := Y \quad 1/2 \oplus \quad Z) \]

\[ B \quad (s := X \quad \cap \quad Y) \quad 1/2 \oplus \quad (s := X \quad \cap \quad Z) \]

The automatic refutation procedure (\textit{Jackal}+\textit{Yices}) for the claimed refinement \( A \subseteq B \) produces the line shown (or similar), which is \( 7x + 3y + 12z = 6 \). The refutation certificate is tuple \( (7, 3, 12 \mid 6) \).
Source-level refutation

\[ A \quad s := X \sqcap (s := Y_{1/2} \oplus Z) \]

\[ B \quad (s := X \sqcap Y)_{1/2} \oplus (s := X \sqcap Z) \]

The automatic refutation procedure (Jackal+Yices) for the claimed refinement \( A \sqsubseteq B \) produces the line shown (or similar), which is \( 7x + 3y + 12z = 6 \). The refutation certificate is tuple \((7, 3, 12 \mid 6)\).
Source-level refutation

A \hspace{1cm} s := X \sqcap (s := Y \frac{1}{2} \oplus Z)

B \hspace{1cm} (s := X \sqcap Y) \frac{1}{2} \oplus (s := X \sqcap Z)

7x + 3y + 12z = 6
[X] abbreviates [s=X] etc

wp.A.certificate

= wp.A.(7[X] + 3[Y] + 12[Z])

= 7 \min (3/2 + 12/2) \hspace{1cm} pGCL programming logic

= 7 \hspace{1cm} \text{arithmetic}
Source-level refutation

\[
A \quad s := X \sqcap (s := Y \frac{1}{2} \oplus Z)
\]

\[
B \quad (s := X \sqcap Y) \frac{1}{2} \oplus (s := X \sqcap Z)
\]

\[
7x + 3y + 12z = 6
\]

[X] abbreviates \([s=X]\) etc

\[
wp.A.\text{certificate}
\]

\[
= wp.A.(7[X] + 3[Y] + 12[Z])
\]

\[
= 7 \min (3/2 + 12/2) \quad \text{pGCL programming logic}
\]

\[
= 7 . \quad \text{arithmetic}
\]
Source-level refutation

\[ A \quad s := X \sqcap (s := Y \frac{1}{2} \oplus Z) \]
\[ B \quad (s := X \sqcap Y) \frac{1}{2} \oplus (s := X \sqcap Z) \]

\[ 7x + 3y + 12z = 6 \]
[X] abbreviates [s=X] etc

wp.Aertificate

\[ = \quad wp.A.(7[X] + 3[Y] + 12[Z]) \]
\[ = 7 \min (3/2 + 12/2) \quad pGCL \text{ programming logic} \]
\[ = 7. \quad \text{arithmetic} \]
Source-level refutation

\[ A \quad s := X \quad \sqcap \quad (s := Y)_{1/2} \oplus Z \quad 7 \]

\[ B \quad (s := X \sqcap Y)_{1/2} \oplus (s := X \sqcap Z) \]

\[ 7x + 3y + 12z = 6 \]

\[ \text{wp.} B. \text{certificate} \]
\[ = \quad \text{wp.} B. (7[X] + 3[Y] + 12[Z]) \]
\[ = \quad (7 \text{ min } 3)/2 + (7 \text{ min } 12)/2 \quad pGCL \text{ logic} \]
\[ = \quad 5 \quad \text{ arithmetic} \]
Source-level refutation

\[ A \quad s := X \quad \sqcap \quad (s := Y \quad \frac{1}{2} \oplus Z) \quad 7 \]

\[ B \quad (s := X \sqcap Y) \quad \frac{1}{2} \oplus \quad (s := X \sqcap Z) \quad 5 \]

\[ 7x + 3y + 12z = 6 \]

\[ wp.B.certificate \]
\[ = \quad wp.B.(7[X] + 3[Y] + 12[Z]) \]
\[ = \quad (7 \min 3)/2 + (7 \min 12)/2 \quad \text{pGCL logic} \]
\[ = \quad 5 . \quad \text{arithmetic} \]
Source-level refutation

\[ A \quad s := X \quad \cap \quad (s := Y_{1/2} \oplus Z) \quad \frac{7}{6} \]

\[ B \quad (s := X \cap Y)_{1/2} \oplus (s := X \cap Z) \quad \frac{5}{6} \]

\[ 7x + 3y + 12z = 6 \]

\[ \text{wp.}B.\text{certificate} \]
\[ = \text{wp.}B.(7[X] + 3[Y] + 12[Z]) \]
\[ = (7 \text{ min } 3)/2 + (7 \text{ min } 12)/2 \quad pGCL \text{ logic} \]
\[ = 5. \quad \text{arithmetic} \]
Source-level refutation

\[ 7x + 3y + 12z = \]

\[ wp.B.\text{certificate} \]

\[ = wp.B.(7[X] + 3[Y] + 12[Z]) \]

\[ = (7 \text{ min } 3)/2 + (7 \text{ min } 12)/2 \]

\[ = 5. \]

Payoff.

For refinement, this “precondition” must not decrease.
For refinement, this “precondition” must not decrease.

A single such failure refutes the whole claimed refinement.
A \quad s := X \sqcap (s := Y_{1/2} \oplus Z) \\
B \quad (s := X \sqcap Y)_{1/2} \oplus Z

\begin{align*}
7x + 3y + 12z &= 6 \\
wp.B &= \frac{1}{2} \left( \frac{7}{5} \right) + \frac{1}{2} \left( \frac{6}{5} \right)
\end{align*}

For refinement, this "precondition" must not decrease.

A single such failure refutes the whole claimed refinement.

Thus \( B \) cannot be a refinement of \( A \).
Source-level refutation

\[
A \quad s := X \cap (s := Y \frac{1}{2} \oplus Z) \quad 7
\]

\[
B \quad (s := X \cap Y) \frac{1}{2} \oplus (s := X \cap Z) \quad \frac{6}{5}
\]

Thus we are able to refute \( A \sqsubseteq B \) at the source level, using the automatically generated probabilistic postcondition

\[7[X] + 3[Y] + 12[Z].\]

For standard programs, we need \( wp.A.post \Rightarrow wp.B.post \) for all postconditions \( post \). For probabilistic programs the analogue is that we must have \( wp.A.post \leq wp.B.post \) for all postconditions \( post \); and thus \( 7 > 6 > 5 \) whence \( 7 \not\leq 5 \) shows that \( A \not\sqsubseteq B \). These \( wp \)-calculations for non-looping programs can be performed automatically, as well, by \( pGCL\text{-}\text{hol}^* \).

* McIver, Hurd and Celiku.
Summary

Given are two (small, loop-free) probabilistic programs $A, B$. Is it true that $A \sqsubseteq B$ for a given initial state? To find out automatically,

1. Submit the source code of $A, B$ to *The Jackal*. From the initial state it forward-generates a finite set of result distributions for $A$, and for $B$.

2. Submit those result sets to *Yices* and check for refinement. If “Yes,” keep the affirmation certificate. If “No,” keep the refutation certificate and...

3. Submit the programs’ texts and the refutation to *pGCL-HOL*, letting it confirm independently the refinement failure at the *source level*.

4. (Use *backward, logic-based reasoning* to find errant code in $B$.)
Summary

Given are two (small, loop-free) probabilistic programs $A,B$. Is it true that $A \sqsubseteq B$ for a given initial state? To find out automatically,

1. Submit the source code of $A,B$ to *The Jackal*. From the initial state it forward-generates a finite set of result distributions for $A$, and for $B$.

2. Submit those result sets to *Yices* and check for refinement. If “Yes,” keep the affirmation certificate. If “No,” keep the refutation certificate and...

3. Submit the programs’ texts and the refutation to *pGCL-HOL*, letting it confirm independently the refinement failure at the source level.

4. (Use backward, logic-based reasoning to find errant code in $B$.)
0. Given a complex distributed large, looping probabilistic system $X$, use $pKAT$ to reason algebraically about it, producing a smaller and simpler no-more-correct “separation/reduction” $Y \sqsubseteq X$ with provisos that are... ...small, loop-free postulated-refinement pairs $(A,B)$. 

**Summary**

Given are two (small, loop-free) probabilistic programs $A,B$. Is it true that $A \sqsubseteq B$ for a given initial state? To find out automatically,

1. Submit the source code of $A,B$ to The Jackal. From the initial state it forward-generates a finite set of result distributions for $A$, and for $B$.

2. Submit those result sets to Yices and check for refinement. If “Yes,” keep the affirmation certificate. If “No,” keep the refutation certificate and...

3. Submit the programs’ texts and the refutation to pGCL-hol, letting it confirm independently the refinement failure at the source level.

4. (Use backward, logic-based reasoning to find errant code in $B$.)
0. Given a complex distributed large, looping probabilistic system $X$, use *pKAT* to reason algebraically about it, producing a smaller and simpler no-more-correct “separation/reduction” $Y \subseteq X$ with provisos that are... ...small, loop-free postulated-refinement pairs $(A,B)$.

**Summary**

Given are two (small, loop-free) probabilistic programs $A, B$. Is it true that $A \sqsubseteq B$ for a given initial state? To find out **automatically**,

1. Submit the source code of $A, B$ to *The Jackal*. From the initial state it forward-generates a finite set of result distributions for $A$, and for $B$.
2. Submit those result sets to *Yices* and check for refinement. If “Yes,” keep the affirmation certificate. If “No,” keep the refutation certificate and...
3. Submit the programs’ texts and the refutation to *pGCL-HOL*, letting it confirm independently the refinement failure at the source level.
4. (Use **backward, logic-based reasoning** to find errant code in $B$.)

*pKA —* McLver, Cohen, *later*

Meinicke, Solin

*pKAT —* Kozen, Aboul-Hosn, *then* McLver, Gonzalia
Summary

0. Given a complex distributed large, looping probabilistic system $X$, use $pKAT$ to reason algebraically about it, producing a smaller and simpler no-more-correct “separation/reduction” $Y \subseteq X$ with provisos that are... small, loop-free postulated-refinement pairs $(A,B)$.

$pKAT$ — Kozen, Aboul-Hosn, then McIver, Gonzalia

$pKA$ — McIver, Cohen, later Meinicke, Solin

Lipton
Back, Sere
von Wright
0. Given a complex distributed large, looping probabilistic system \( X \), use \( pKAT \) to reason algebraically about it, producing a smaller and simpler no-more-correct “separation/reduction” \( Y \subseteq X \) with provisos that are... ...small, loop-free postulated-refinement pairs \((A,B)\).

Summary

Lipton
Back, Sere
von Wright

\( pKA \) — McLver, Cohen, later Meinicke, Solin

\( pKAT \) — Kozen, Aboul-Hosn, then McLver, Gonzalia
Given a complex distributed large, looping probabilistic system $X$, use pKAT to reason algebraically about it, producing a smaller and simpler no-more-correct "separation/reduction" $Y \sqsubseteq X$ with provisos that are... small, loop-free postulated-refinement pairs $(A, B)$.
Refutation at source-code level
Refutation at source-code level
Refutation at source-code level

- Jackal
- Yices
- pGCL

B's source code

B's polytope

interpolation
affirmation
certificate


A's source code

A's polytope

hyperplane
refutation
certificate
Higher dimensions...
Higher dimensions...

A polytope: \( p_w \leq 2p_x, 2p_y, 2p_z \) etc
Higher dimensions...

A polytope: $p_w \leq 2p_x, 2p_y, 2p_z$ etc
Higher dimensions...

A polytope: \( p_w \leq 2p_x, 2p_y, 2p_z \text{ etc} \)
Higher dimensions...

refuting hyperplane

refutable point

A polytope: \( p_w \leq 2p_x, 2p_y, 2p_z \) etc
Higher dimensions...

refuting hyperplane

refutable point

A polytope: $p_w \leq 2p_x, 2p_y, 2p_z$ etc

automatically generated refuting normal, for source-level analysis
$B$ is “1 if $B$ else 0.”

$\text{prog}_1 \ [B] \oplus \text{prog}_2$

$\text{prog}_1 \ \text{if} \ B \ \text{else} \ \text{prog}_2$
Enter (or re-enter) the loop with probability $p$, which can be an expression.

\[ \text{do } p \rightarrow \text{ prog od} \]
pGCL semantics

\[ \text{do } p \rightarrow \text{prog od} \]

is as usual \( (\mu X \cdot (\text{prog}; X)_{p \oplus \text{skip}}) \)

\[ \text{do } [B] \rightarrow \text{prog od} \]

while \( B \) do \( \text{prog od} \)
# Definition of terms

## Unbounded
- **Affirmation complete**: Very hard in general.
- **Affirmation sound**: Theorem provers do it.
- **Refutation complete**: Very hard in general.
- **Refutation sound**: Model checkers do it.

## Bounded (for model checking)
- **Affirmation complete**: No false alarms.
- **Affirmation sound**: No false ok’s.
- **Refutation complete**: No false ok’s.
- **Refutation sound**: No false alarms.