LMS/BCS-FACS Evening Seminar — London and Sydney.

Probabilistic Datatypes: Automating verification for abstract probabilistic reasoning

Annabelle McIver Chris Chen Carroll Morgan

❖ Probabilistic power domains (Jones, Plotkin, Saheb-Djahromi)

- ❖ Source-level reasoning for sequential programs (no non-determinism) (Kozen) 1990-ish
- ❖ Source-level reasoning for sequential programs, pGCL (yes non-determinism!)
- Randomised algorithms for resource, security, performance.
	- 1980-ish
- Efficient reasoning principles, semantics, algebras, logics.
	-

❖ Probabilistic power domains (Jones, Plotkin, Saheb-Djahromi)

- ❖ Source-level reasoning for sequential programs (no non-determinism) (Kozen)
	- 1990-ish
- ❖ Source-level reasoning for sequential programs, pGCL (yes non-determinism!)
- Randomised algorithms for resource, security, performance.
	- 1980-ish
- Efficient reasoning principles, semantics, algebras, logics.
	-

❖ Probabilistic power domains (Jones, Plotkin, Saheb-Djahromi)

- ❖ Source-level reasoning for sequential programs (no non-determinism) (Kozen) 1990-ish
- ❖ Source-level reasoning for sequential programs, pGCL (yes non-determinism!)
- Randomised algorithms for resource, security, performance.
	- 1980-ish
- Efficient reasoning principles, semantics, algebras, logics.
	-

❖ Probabilistic power domains (Jones, Plotkin, Saheb-Djahromi)

- ❖ Source-level reasoning for sequential programs (no non-determinism) (Kozen)
	- 1990-ish
- ❖ Source-level reasoning for sequential programs, pGCL (yes non-determinism!)
- Randomised algorithms for resource, security, performance.
	- 1980-ish
- Efficient reasoning principles, semantics, algebras, logics.
	-

Efficient Reasoning for Probabilistic Programs

❖ Non-determinism — enables transition abstraction. $P \sqsubseteq Q$ means that program Q satisfies all the properties of P

❖ Source-level reasoning for sequential programs, pGCL (yes non-determinism!)

❖ Source-level reasoning. $\{\phi\}$ **P** $\{\psi\}$ is a *Probabilistic* Hoare-style triple; ϕ and ψ are real-valued functions.

❖ Tools for verifying all the triples…automatically if possible!

A challenge program

$$
\{ 1/N \} # \text{Precondition} \\\ \text{var c, } v = 0, 1 \\\ \text{while}(v < N \text{ or } c \geq N) \} \\\ \{ \text{Inv} \times [v < N \text{ or } c \geq N] \} \\\ \Box (v \leq N) \rightarrow v = 2v \\\ \ C = 2c \quad \text{if } v < N \} \rightarrow v, c = v - N, c - N \\\ \{ \text{Inv} \} \\\ \ \{ \text{C = i]} \} \# \text{Post condition for an}
$$

Set c uniformly between 0 and N-1. If the invariant S

The invariant Invari
The invariant Invari

It turns out that, Inv is non-zero with seemingly unrelated probabilities for ap-

cannot be expressed because it is too complex, the technique of in-technique of in-technique of in-technique of in-

- ❖ Why is this fast?
- ❖ Why is this a challenge?
- ❖ What is Inv ?
- ❖ What should we do?

A challenge program, N=5, i=3

A challenge program, N=5, i=3

cannot be expressed because it is too complex, this makes the technique of in-

A challenge program, N=5, i=3

What should we do?

TEST P

Abstract datatypes

The motivation behind the work in very-high-level languages is to ease the programming task by providing the programmer with a language containing primitives or abstractions suitable to his problem area. The programmer is then able to spend his effort in the right place; he concentrates on solving his problem, and the resulting program will be more reliable as a result. Clearly, this is a worthwhile goal.

Unfortunately, it is very difficult for a designer to select in advance all the abstractions which the users of his language might need. If a language is to be used at all, it is likely to be used to solve problems which its designer did not envision, and for which the abstractions embedded in the language are not sufficient.

> ❖ Barbara Liskov and Stephen Zilles, Proceedings of the ACM SIGPLAN symposium on Very high level *[languages](https://dl.acm.org/doi/proceedings/10.1145/800233)*, 1974

<u>Abstract</u>

An example: a quick prototype Thus we will have to change the abstract Γ introduced the abstract Set specification slightly, and we introduce Δ n evomple: a guitek prototome a sind oxiding die die rent specification, we in our place of place. the data-type invariant for the data-type invariant for the class with "Dti" (instead of the "Inv" we use for
Invariant for loops

class Set(N): # New abstract version, capacity limited. # DTI: $|ss| \le N$ # Data-type invariant. local: # PRE: N>=0 # Data-type precondition. $ss = \{\}$ def makeEmpty: $ss = \{\}$ def add(s): PRE: $|ss| := N$ # Must hold when add() is called. $ss=$ ss \cup {s} def isIn(s): return $s \in ss$

representation of it.

Taken from: Formal Methods, Informally, Carroll Morgan (to appear, CUP). The second new feature is the precondition Pre following the local keyword. ¹

Programming with specifications Thus we will have to change the abstract Set specification slightly, and we introduce some new features and it is the abstract of the abstract data-type, we have abstract of the abstract data-type, we have abstract of the abstract data-type, an a slightly dierent specification, written out all together in one place. We indicate

class $Set(N):$ # New abstract version, capacity limited. # Dti: |ss|<=N # Data-type invariant.

local: $ss = \{\}$ def makeEmpty: $ss = \{\}$ def add(s): $ss=$ $ss \cup \{s\}$

the data-type invariant for the class with the class with the class with "Dti" (instead of the "Inv" we use fo
Invariant for loops with the "Invariant" with the "Invariant" with the "Invariant" with the "Invariant" with t

def isIn(s): return $s \in ss$

at its beginning, to reflect its special purpose:

Pre: N>=0 # Data-type precondition.

PRE: $|ss| := N$ # Must hold when add() is called.

(10.1)

SetmySet = Set(10); mySet*.*add(3); mySet*.*add(4); Bool z = mySet*.*isIn(5);

side

What does this mean for verification? operation is a strategied with what we have a with what a with what a with ward wi What does this mean far warificat **if any and the data is a decomposition of the component of the compo**

Definition 1. *An* encapsulated datatype *is a triple* (*I, OP, F*)*, where I and F* are two distinguished operations called respectively the initialisation and finali*sation, and OP is an indexed set of publicly accessible operations. [12]* that *P* v *P*⁰ meaning (as stated above) desired behaviours of *P* are preserved **)** efinit

the concrete implementation is run.

Definition 2. *A datatype* (I, OP, F) *is* refined by (I', OP', F') [12] if, for every *program P expressible using the constructs mentioned above, including calls on corresponding operators in OP and OP', we have*

 $I; \mathcal{P}(OP); F \subseteq I'; \mathcal{P}(OP'); F'$,

all its desired properties preserved, we say that (*I, OP, F*) is *itself* refined by

combination of probability *and* nondeterminism for datatypes [27]; in the next

where " ;*" indicates sequential composition.*

Today's talk

❖ Can we still use the abstract specification to prove properties of programs that use probabilistic

- ❖ What happens when some of the behaviour can be probabilistic?
- ❖ Do the traditional proof methods (eg simulation) still work?
- datatypes?
- ❖ If they don't, what must be changed?
	- ❖ A semantics and refinement that distinguishes hidden and visible state;
	- ❖ Does it apply to our challenge program?
	- ❖ We need to talk about information leaks…

What happens when the datatype is probabilistic?

datatype (*IA*, *{*Flip*A}*, *FA*) to be refined by (*I^C* , *{*Flip*^C }*, *F^C*) accord-Flip*^C* : coin:= c;

- var coin # Local variable
- Flip*A*: # Flip on demand. coin:= H $1/2 \oplus$ coin:= T; return coin
	-
	- (a) The abstract datatype

Probabilistic choice

Abstract (specification) datatype for a coin flip $if constant is not a constant.$ Allealion, dalai

the question here is whether the encapsulation, guaranteeing that vari-

Programming with the coin datatype ing to Def. 2, given a definition of the observation of the observation of the observation of the observation assumption into account. I rogramming with the computation

var coin # Local variable *IA*: skip Flip*A*: # Flip on demand. coin:= H $_{1/2} \oplus$ coin:= T; return coin *FA*: skip

(a) The abstract datatype

Fig. 1: Abstract and concrete probabilistic datatypes

involvement is a the point *in the Flipp* set *randomly in Flipp* set F *exactly the same way as* Flip*^A* did. In fact, as shown in [27], the operation *rep* given by c:= 0 ¹*/*2 1 satisfies (1)–(3) for the datatypes in Fig. 1a and Fig. 1b. choice nondeterministic

As what is the problems where the two data two datastic theorems where the two datastic transitions of the two data types of the two datastic transitions of the two datastic transitions of the two data types of two datasti seemingly *are* distinguished when we define refinement using the standard sechoice What is the probability that $g=v$ finally?

$v:= H \sqcap T$; g:= Flip() # 1⁶³

skip # *F^A* inlined.

skip # *F^A* inlined.

 $\frac{F}{\sqrt{2}}$. $\frac{F}{\sqrt{2}}$ This is what would be expected, but can we prove it formally? What happens when we implement the ac datatype?

I $v := H \sqcap T$; $g := \text{FIip}()$

Reasoning with the coin datatype **Examing with the coin data**

skip # *F^C* inlined.

What is the probability that $g=v$ finally?

1/2 because the flip comes after the nondeterminism….?

```
var coin # Local variable
IA: skip
FlipA: # Flip on demand.
         coin:= H
     _{1/2} \oplus coin:= T;
     return coin
FA: skip
             \frac{1}{\sqrt{2}}
```
c, then immediately "re-flips" c, ready for interesting the source of its next use. And so its next use. And s
And so its next use of its next use of its next use. And so its next use of its next use. And so its next use

(a) The abstract datatype we

Fig. 1: Abstract and concrete probabilistic datatypes versus (*I^C* , *{*Flip*^C }*, *F^C*). Inlining in the abstract Fig. 2a, the flip is made into local variable coin but then assigned immediately to the global variable g via Flip*A*'s return coin statement. In Fig. 2b however it is the earlier coin flip into c, carried out during the initialisation *I^C* , that Flip*^C* assigns to g. F

datatype (*IA*, *{*Flip*A}*, *FA*) to be refined by (*I^C* , *{*Flip*^C }*, *F^C*) accord-

Flip*^C* : coin:= c;

semantics determines whether or not this verification technique is valid within \blacksquare Consider the user's program in Fig. 2c that sets a global variable v nondeter $m_{\rm m}$ to H or H or H or H tically via the Flip operation. (That is, neither g nor v are in the encapsulation.) The program is an example of a context *P*(*·*) in Def. 2, and so it is reasonable to F. T^D() v:= H u T; g:= Flip() **A Particular Parties** $g := \text{Flip}()$ $\left\{\begin{array}{c} \text{Fing} \\ \text{Fing} \end{array}\right\}$

(b) Inlined concrete encapsulation probability?What about

Fig. 2a and Fig. 2b.

$\{ Post Condition \}$ \int Post Condition tion}

\mathbf{I} *I* \mathbf{v} g:= Flip() # Is this Flip*A*, or Flip*^C* ? F and F and F and F is matter. $v := H \sqcap T;$ v:= H u T;

ing to Def. 2, given a definition of refinement that takes the observability skip $\{Pre Con$ (a) Indianagement absolute the Condition { Pre Condition}

skip # *F^A* inlined.

We can reason about the program by using the abstract datatype and Hoare Triples... \mathbf{v}_I w c can reason about the program by using the abstr about the program by using the g:= H ¹*/*2 T; # Flip*A*. vand Hoare Triples g:= H ¹*/*2 T; # Flip*A*. ason about the program by the g:= c; c:= H 1^{/2} H 1^{/2} T ¹/2 T ¹ type and rioale ruples... (a) Inlined abstract encapsulation (a) Indian to the program by using the epstract determented Heare Triples

I

We can reason about the program by using the abstract datatype and pGCL... W_a and possess v:= H u T; # Program. 11 the stream proprem 2, then 0 vertical values of \mathbb{C}^n g:= H ¹*/*2 T; # Flip*A*. reason about the program b r dataly pc and perul_i... (a) Inlined abstract encapsulation on obout the precrem by using the ebstract determe and pCCI

var coin # Local variable *IA*: skip Flip*A*: # Flip on demand. coin:= H $1/2 \oplus$ coin:= T; return coin skip and the set of $\{p\}$ $\begin{array}{ccc} \texttt{pmand.} \end{array}$ $\{p\}$

invocation of Flip*^C* , and at that point *it is as though* coin *is set randomly in exactly the same way as* Flip*^A* did. In fact, as shown in [27], the operation *rep* $\{[g=v]\}$ having opposite values. $W_{\rm eff}$ is the probability that is the probability that values of μ is the probability that values of μ $\int \sigma - v \, dt$

REFINEMENT AND PROOF FOR FIIS SYSTEMS

datatype (*IA*, *{*Flip*A}*, *FA*) to be refined by (*I^C* , *{*Flip*^C }*, *F^C*) accord-

FA: skip

(a) The abstract datatype

 $wp.Prog.[g=v] = p$ setting of c used later to assign to assign to assign to assign to assign to assign to a resolved after initial
The coin has already been resolved after initial after initial after initial after initial after initial after \mathbb{F} is the probability that is the probability of \mathbb{F}

$\{ p \}$

I^C : c:= H ¹*/*2 T # Pre-flip.

sation, there is no way for the calling program until the calling program until the calling program until the c
The calling program until the calling program until the calling program until the calling program until the ca versus (*I^C* , *{*Flip*^C }*, *F^C*). Inlining in the abstract Fig. 2a, the flip is made into local variable coin but then assigned immediately to the global variable g via Flip*A*'s return coin statement. In Fig. 2b however it is the earlier coin flip into c, carried out during the initialisation *I^C* , that Flip*^C* assigns to g.

 $\alpha \cdot - \Gamma$ istic datatypes whether or not this verification technique is valid within the Γ \blacksquare Consider the user's program in Fig. 2c that sets a global variable v nondeter $m_{\rm m}$ to H or H or H or H tically via the Flip operation. (That is, neither g nor v are in the encapsulation.) The program is an example of a context *P*(*·*) in Def. 2, and so it is reasonable to F. T^D() v:= H u T;

Fig. 2a and Fig. 2b.

I \mathbf{v} g:= Flip() # Is this Flip*A*, or Flip*^C* ? F and F and F and F is matter. $v := H \sqcap T;$ v:= H u T;

skip # *F^A* inlined.

(c) User's code

I

(c) User's code

able c *cannot be accessed or observed* by the calling program, suces for

c:= H ¹*/*2 T # *I^C* inlined.

We can reason about the program by using the abstract datatype C abstiact data von about the program! gian de dit die program d hetract datatyne pou accuracy po g:= H ¹*/*2 T; # Flip*A*. skip # *F^A* inlined. g:= c; c:= H ¹*/*2 T # Flip*^C* . re abstract dataly pe skip # *F^A* inlined. $\mathbf{1}$ or $\mathbf{0}$ absorption absorption $\mathbf{0}$ skip # *F^C* inlined. egen shout the program by uging the shatract detation

c:= H ¹*/*2 T # *I^C* inlined.

We can reason about the program by using the abstract datatype von about the program! gian de dit die program d hetract datatyne pou accuracy po g:= H ¹*/*2 T; # Flip*A*. skip # *F^A* inlined. g:= c; c:= H ¹*/*2 T # Flip*^C* . re abstract dataly pe skip # *F^A* inlined. $\mathbf{1}$ or $\mathbf{0}$ absorption absorption $\mathbf{0}$ skip # *F^C* inlined. egen shout the program by uging the shatract detation datatype (*IA*, *{*Flip*A}*, *FA*) to be refined by (*I^C* , *{*Flip*^C }*, *F^C*) accordic abstract dataly pe

the question here is whether the encapsulation, guaranteeing that vari-

assumption into account.

- v:= H u T; # Program. g:= H ¹*/*2 T; # Flip*A*. skip # *F^A* inlined. rii *I'*n 1 g:= c; c:= H ¹*/*2 T # Flip*^C* . skip # *F^C* inlined. $[H=T]/2 + [T=T]/2$ $F = T / 2$ ${[\text{H=H}]/2 + [\text{T=H}]/2 \text{ MIN}}$
	- *I* \mathbf{v} g:= Flip() # Is this Flip*A*, or Flip*^C* ? F and F and F and F is matter. $v: = H \sqcap T;$
	- (c) User's code Fig. 2: Using the copy rule the operation's text at its point of call (taking care with variable capture), then using normal refinement rules on the resulting operation-call -free programs. $=$ v]/2 + [T=v]/ $[v=0]$ $[v=0$ ${[H=v]/2 + [T=v]/2}$
		- semantics determines whether or not this verification technique is valid within \blacksquare Consider the user's program in Fig. 2c that sets a global variable v nondeter $m_{\rm m}$ to H or H or H or H tically via the Flip operation. (That is, neither g nor v are in the encapsulation.) The program is an example of a context *P*(*·*) in Def. 2, and so it is reasonable to F. T. D. () g:= Flip() # Is this Flip*A*, or Flip*^C* ? $g :=$ $Flip()$
- comments above concerning the idea that the internals of the datatype are "not observable" wrt. the calling program at run-time, one would hope that there is $\{[g=v]\}$ in Fig. 2a and Fig. 2b, for the instantiation of respectively (*IA*, *{*Flip*A}*, *FA*) g:= Flip() # Is this Flip*A*, or Flip*^C* ?
- versus (*I^C* , *{*Flip*^C }*, *F^C*). Inlining in the abstract Fig. 2a, the flip is made into local variable coin but then assigned immediately to the global variable g via Flip*A*'s return coin statement. In Fig. 2b however it is the earlier coin flip into c, carried out during the initialisation *I^C* , that Flip*^C* assigns to g. F and F and F and F and F is matter.
- 4 In fact the probabilities will be equal, since the same argument applies to the same argument applies $H \sim -1$ W is the probability that is the probability that values \mathbb{R}^n is the probability that values \mathbb{R}^n \mathbf{F} , the copy rule \mathbf{F} and \mathbf{F} are computations of the copy rule \mathbf{F} $\{[g=v]\}$

skip # *F^A* inlined.

c:= H ¹*/*2 T # *I^C* inlined.

We can reason about the program by using the abstract datatype von about the program! gian de dit die program d hetract datatyne pou accuracy po g:= H ¹*/*2 T; # Flip*A*. skip # *F^A* inlined. g:= c; c:= H ¹*/*2 T # Flip*^C* . re abstract dataly pe skip # *F^A* inlined. $\mathbf{1}$ or $\mathbf{0}$ absorption absorption $\mathbf{0}$ skip # *F^C* inlined.

(c) User's code

I

- skip # *I^A* inlined. $\mathbf{v} = \mathbf{v}$ $\frac{1}{2}$ g:= Flip
() # Is this Flip
() # Is this Flip A, or Flip A, o $\{ 1/2 \}$
- *I* \mathbf{v} g:= Flip() # Is this Flip*A*, or Flip*^C* ? F and F and F and F is matter. $v: = H \sqcap T;$
- (c) User's code Fig. 2: Using the copy rule the operation's text at its point of call (taking care with variable capture), then using normal refinement rules on the resulting operation-call -free programs. $=$ v]/2 + [T=v]/ $[v=0]$ $[v=0$ ${[H=v]/2 + [T=v]/2}$
- semantics determines whether or not this verification technique is valid within \blacksquare Consider the user's program in Fig. 2c that sets a global variable v nondeter $m_{\rm m}$ to H or H or H or H tically via the Flip operation. (That is, neither g nor v are in the encapsulation.) The program is an example of a context *P*(*·*) in Def. 2, and so it is reasonable to F. T. D. () g:= Flip() # Is this Flip*A*, or Flip*^C* ? $g :=$ $Flip()$ $g := \text{Flip}()$
- (*I*, *{*Flip*}*, *F*) with either (*IA*, *{*Flip*A}*, *FA*) or (*I^C* , *{*Flip*^C }*, *F^C*). Given the comments above concerning the idea that the internals of the datatype are "not observable" wrt. the calling program at run-time, one would hope that there is $\{[g=v]\}$ in Fig. 2a and Fig. 2b, for the instantiation of respectively (*IA*, *{*Flip*A}*, *FA*) g:= Flip() # Is this Flip*A*, or Flip*^C* ? **F** $\{g=v\}$ $\bm F$
	- versus (*I^C* , *{*Flip*^C }*, *F^C*). Inlining in the abstract Fig. 2a, the flip is made into local variable coin but then assigned immediately to the global variable g via Flip*A*'s return coin statement. In Fig. 2b however it is the earlier coin flip into c, carried out during the initialisation *I^C* , that Flip*^C* assigns to g. F and F and F and F and F is matter.

skip # *F^A* inlined.

c:= H ¹*/*2 T # *I^C* inlined.

We can reason about the program by using the abstract datatype von about the program! gian de dit die program d hetract datatyne pou accuracy po g:= H ¹*/*2 T; # Flip*A*. skip # *F^A* inlined. g:= c; c:= H ¹*/*2 T # Flip*^C* . re abstract dataly pe skip # *F^A* inlined. $\mathbf{1}$ or $\mathbf{0}$ absorption absorption $\mathbf{0}$ skip # *F^C* inlined.

(c) User's code

I

c:= H ¹*/*2 T # *I^C* inlined.

able c *cannot be accessed or observed* by the calling program, suces for

We can reason about the program by using the abstract datatype von about the program! gian de dit die program d hetract datatyne pou accuracy po g:= H ¹*/*2 T; # Flip*A*. skip # *F^A* inlined. g:= c; c:= H ¹*/*2 T # Flip*^C* . re abstract dataly pe skip # *F^A* inlined. $\mathbf{1}$ or $\mathbf{0}$ absorption absorption $\mathbf{0}$ skip # *F^C* inlined. (a) Inlined abstract encapsulation (b) Inlined concrete encapsulation o obotroot dototroo C abstiact data

wp.I.[1/2]

skip # *F^A* inlined. $\{ 1/2 \}$ F and F and F and F is matter. What is the probability that is the probability that values of \blacksquare the operation of call (taking capture), then \mathcal{A} semantics determines whether or not this verification technique is valid within \blacksquare Consider the user's program in Fig. 2c that sets a global variable v nondeter $m_{\rm m}$ to H or H or H or H tically via the Flip operation. (That is, neither g nor v are in the encapsulation.) $g := \text{flip}()$ (*I*, *{*Flip*}*, *F*) with either (*IA*, *{*Flip*A}*, *FA*) or (*I^C* , *{*Flip*^C }*, *F^C*). Given the comments above concerning the idea that the idea that the idea that the idea that the internals of the datatype observable" wrt. the calling program at run-time, one would hope that there is *I I* $v := H \sqcap T;$ F and F and F and F and F is matter. F and F and F and F and F and F is matter. F and F and F and F and F and F g:= Flip() # Is this Flip*A*, or Flip*^C* ? F^*

c, carried out during the initialisation *I^C* , that Flip*^C* assigns to g. If refinement from abstract to concrete *does* hold, then the probability that v and g have the same value finally in Fig. 2b showld be at least that for Fig. 2a. 4 $\{ [g=v] \}$

v:= H u T; # Program.

g:///2 Times.org/
The thickness was the set of skip # *F^A* inlined. g:= H ¹*/*2 T; # Flip*A*. skip # *F^A* inlined. g:= H ¹*/*2 T; # Flip*A*. skip # *F^A* inlined. g:= H ¹*/*2 T; # Flip*A*. **Solution 4 Windows**

(a) Inlined abstract encapsulation

(a) Inlined abstract encapsulation of the state of the sta
In the state of the

ar coin # Local variable *IA*: skip Flip*A*: # Flip on demand. coin:= H $1/2 \oplus$ coin:= T; return coin *FA*: skip (a) The abstract datatype

Why is this valid?

Justification using the "copy rule"…

able c *cannot be accessed or observed* by the calling program, suces for

given by c:= 0 1/2 1 satisfies (1)–(3) for the data types in Fig. 1 satisfies (1)–(3) for the data types in Fig. 1

The copy rule: inlining the code should mean the same thing m_{F} rule: inlining the code chould mean the came thing all the same t var coin # Local variable var
Local variable varia var coin,c # Local variables inlining the code should mean the same thing *Fhe convrule: inlining the code should meet* $F = \frac{1}{2}$ skip # *F^A* inlined. skip # *F^A* inlined. \blacksquare is a strategiment absorption of the strategiment absorption \blacksquare skip # *F^C* inlined.

able c *cannot be accessed or observed* by the calling program, suces for

g:= H ¹*/*2 T; # Flip*A*.

g:= c; c:= H ¹*/*2 T # Flip*^C* .

g:= H ¹*/*2 T; # Flip*A*.

g:= c; c:= H ¹*/*2 T # Flip*^C* .

How does all of this work with refinement of datatypes?

A refinement example datatype (*IA*, *{*Flip*A}*, *FA*) to be refined by (*I^C* , *{*Flip*^C }*, *F^C*) according the definition of the observability of refining that the observability of refining that the observability o assumption into account.

able c *cannot be accessed or observed* by the calling program, suces for

var coin # Local variable

IA: skip Flip*A*: # Flip on demand. coin:= H $_{1/2} \oplus$ coin:= T; return coin *FA*: skip (a) The abstract datatype

var coin,c # Local variables I_C : c:= H_{1/2} \oplus T # Pre-flip. Flip_{*C*}: coin:= c; $c:= H_{1/2} \oplus T$; # Pre-flip. return coin

F^C : skip

(b) The concrete datatype

Fig. 1: Abstract and concrete probabilistic datatypes

ITTIV SHOULD BE CONCRETE SHIPS SHOULD BE Why should the concrete refine the abstract?

A refinement example ing to Def. 2, given a definition of refinition of refinition of refinition of refinition of refinition of ref
That takes the observability of refinition of refinition of refinition of refinition of refinition of refinitio A rel

datatype (*IA*, *{*Flip*A}*, *FA*) to be refined by (*I^C* , *{*Flip*^C }*, *F^C*) accord-

var coin # Local variable

IA: skip Flip*A*: # Flip on demand. coin:= H $_{1/2} \oplus$ coin:= T; return coin *FA*: skip (a) The abstract datatype

(b) The concrete datatype

Fig. 1: Abstract and concrete probabilistic datatypes

 $\frac{1}{\sqrt{1-\frac{1$ setting of current to assign to a coincil the coincil of coincil of the coincil of the coincil of the coincil o The abstract datatype uses a "hidden" variable c to store a pre-flipped value — why shouldn't that matter?

sequential composition, assignment, branching and loops, and also both demonic

Remember the definition of refinement? rcomention the uchinition of femicine for the next of the next of the next of α section we review some known results with an example. emember the definition of refinement? v:= H u T; # Program.

Definition 2. A datatype (I, OP, F) is refined by (I', OP', F') [12] if, for every program P expressible using the constructs mentioned above, including calls on *program P expressible using the constructs mentioned above, including calls on corresponding operators in OP and OP', we have* $\emph{rresponding operators in OP and O}$

There is a wealth of literature is a wealth of literature on verification methods for proving refinements for p
There is a wealth of literature on verification methods for proving refinements for proving refinements of the

where " ;*" indicates sequential composition.*

choice cannot be resolved *the following infermal* $\overline{}$ *OP^j* ; *rep* v *rep*; *OP* ⁰ Demonic nondeterministic on the basis of internal state that it cannot "access" or "observe"

 $I; \mathcal{P}(\mathit{OP}); F \subseteq I'; \mathcal{P}(\mathit{OP}'); F'$, $I; P(OP); F = I; P(OP'); F$

 $v:= \text{H} \sqcap \text{T};$
g:= Flip()

of datatypes (Def. 2). A common method is *simulation*:

F v *rep*; *F*⁰ (3)

c nondeterministic *I Properties and in particular whether or* (*I, OP*, *In particular whether or* $\mathbf{v} := \mathbf{H} \cap \mathbf{T}$ and there has been a *I*, *rep i rep i rep I i i not* available, even at *j* 8*j*²*j***₂***J***₂^{***j***}₂***J***₂^{***j***}₂***J***₂^{***j***}₂***J***₂^{***j***}₂***J***₂^{***j***}₂***J***₂^{***j***}₂***J***₂^{***j***}** $g := \text{Flip}()$ supp of a local variable is supp F $\frac{100 \text{ cm}}{m}$ $\frac{m}{m}$ not there has been a "preflip" of a local variable is "run-time"

v:= H u T; # Program.

I

 \int_{C} $\frac{1}{2}$ \int_{C} $\frac{1}{2}$ *IC* : c:= H_{1/2} \oplus T # Pre-flip. Flip*^C* : coin:= c;

 $c:= H_{1/2} \bigoplus T;$ c:= H_{1/2} OT; # Pre-f['] hj return coin

(a) India abstract encapsulation abstract encapsulation abstract encapsulation abstract encapsulation abstract
India abstract encapsulation abstract en la provincia abstract en la provincia abstract en la provincia abstra (b) The concrete datatype

var coin # Local variable	var coin, c # Local
I_A : skip	I_C : $C:=H_{1/2}\oplus T$ #
Flip_A : # Flip on demand.	Flip_{C} : coin:= c;
$coin := H$	$c := H_{1/2} \oplus T$;
$_{1/2}\oplus$ coin:= T;	return coin
return coin	
F_A : skip	F_C : skip
(a) The abstract datatype	(b) The concrete of

Fig. 1: Abstract and concrete probabilistic datatypes

atypes should therefore be	I
in terms of their behaviour,	$v := H \sqcap T$;
one should result in g and v	$g := F \text{Lip}(t)$

setting of coincident to assign the coincident of the coincident in the coinci equivalent in terms of their behaviour, $\overline{v := \mathbf{H} \cup \mathbf{H}}$ $\mathbf{g} := \text{Flip}()$ *exactly the same way as* Flip*^A* did. In fact, as shown in [27], the operation *rep* gequal with probability $1/2$ and $\frac{1}{2}$ These datatypes should therefore be and either one should result in g and v being equal with probability 1/2

But is this enough to prove refinement to prove refinement to prove refinement to prove refinement to prove re
In this enough to prove refinement to prove refinement to prove refinement to prove the construction to the pro

Remember the definition of refinement? ing to Def. 2, given a definition of refinition of α nemher th

datatype (*IA*, *{*Flip*A}*, *FA*) to be refined by (*I^C* , *{*Flip*^C }*, *F^C*) accord-

var coin,c # Local variables

Let's look again at the copy rule, this time for the concrete datatype (a) a comparative material this time for the concrete datation α at the copy rule, this time for the concrete datatype \mathbf{I} Inlined abstract encapsulation \mathbf{I} in the concrete encapsulation \mathbf{C} (a) Inlined abstract encapsulation (b) Inlined concrete encapsulation of the state encapsulation of the state encapsulation of the state encapsula
In the state encapsulation of the state encapsulation of the state encapsulation of the state encapsulation of (a) Inlined abstract encapsulation (b) Inlined concrete encapsulation of the concrete encapsulation of the concrete encapsulation of the concrete
In the concrete encapsulation of the concrete encapsulation of the concrete encapsulation of the concrete enca
 ing to Def. 2, given a definition of refinement that takes the observability any rule, this tin

If refinement from abstract to concrete *does* hold, then the probability that v and g have the same value finally in Fig. 2b should be at least that for Fig. 2a. ⁴ $\mathcal{S} = \{ \mathcal{S} \mid \mathcal{S} \}$ and the program framework interpret the program fragments in

 \mathcal{A} in fact the probabilities will be equal, since the same argument applies to the same argument ap

tics for datatypes? semantics for datatypes? Fig. 2: Using the copy rule of the copy rule
The copy rule of the cop IS PGCL/MDP the wrong hidden state is revealed! Is pGCL/MDP the wrong semantics for datatypes?

(a) Inlined abstract encapsulation

datatype (*IA*, *{*Flip*A}*, *FA*) to be refined by (*I^C* , *{*Flip*^C }*, *F^C*) accord-

What is the right semantics for probabilistic datatypes?

Let's take another look at refinement **production of probability and non-determinism for data terminism** for data terminism for data terminism for data

Definition 2. *A datatype* (I, OP, F) *is* refined by (I', OP', F') [12] if, for every *program P expressible using the constructs mentioned above, including calls on corresponding operators in OP and OP', we have*

sequential composition, assignment, branching and loops, and also both demonic

section we review some known results with an example.

where " ;*" indicates sequential composition.*

There is a wealth of literature on verification methods for proving refinement

 $I; \mathcal{P}(\mathit{OP}); F \subseteq I'; \mathcal{P}(\mathit{OP}'); F'$,

of datatypes (Def. 2). A common method is *simulation*:

OP^j ; *rep* v *rep*; *OP* ⁰

^j 8*j*2*J* (2)

Definition 3. *We say that an operation rep is a* simulation *from* (*I, OP, F*) *to* assumption) that the calling program has "no access" to the run-time internals of the datatype. This doesn't matter when there is no *III Ve seen does* when probability is involved. The observed behaviour of datatypes makes an (unarticulated probability, but, as we've seen does when probability is involved.

I;*P*(*OP*); *F* v *I*⁰

Simulation is a well-known technique for this situation. There is a wealth of literature on verification methods for Γ and Γ refinement Γ refinement Γ Simulation is a well-known technique for this situat

;*P*(*OP* ⁰

); *F*⁰

,

of datatypes (Def. 2). A common method is *simulation*:

In order however for simulation (Def. 3) to establish refinement (Def. 2), an

Definition 3. *We say that an operation rep is a* simulation *from* (*I, OP, F*) *to* (I', OP', F') *if -using* $j \in J$ *to index corresponding operations in OP and OP'the following inequations hold [12] :* **Definition 3.** We say that an operation rep is a simulation from (I, OP, F) to $(I'OP'F')$ if $\nexists I$ to index corresponding operations in OP and OP' $the following$ in skip # *F^C* inlined.

I	Simulation should be consistent with
$\mathbf{g} := \mathbf{F} \mathbf{L} \mathbf{I} \mathbf{p}(\mathbf{I})$	Copy rule, using a semantics that the operational assumptions.

g:= Flip() the operational assumptions. Simulation *should be consistent* with a copy rule, using a semantics that reflects

v:= H u T; # Program.

v:= H u T; # Program.

Definition 3. *We say that an operation rep is a* simulation *from* (*I, OP, F*) *to* (I', OP', F') *if -using* $j \in J$ *to index corresponding operations in OP and OP'the following inequations hold [12] :* **Dennition 3.** We say that an operation rep is a simulation $\left(\begin{array}{c} \mu \end{array} \right)$, $\left(\begin{array}{c} \mu \end{array} \right)$ by μ and μ or μ or μ and μ and μ and μ and μ $\mathcal{L}(\mathcal{L}(\mathcal{L}) = \mathcal{L}(\mathcal{L}(\mathcal{L}))$ is a functional field $\mathcal{L}(\mathcal{L})$.

I;*P*(*OP*); *F* v *I*⁰

We can "simulate" concrete behaviours with abstract ones We can "cimulate" concrete behaviours; Friday Constitution of the previous of the previous

;*P*(*OP* ⁰

³ For example, desired properties of sequential programs are often their *termination*

in advance and, after having delivered it, "pre-flips" to be ready for next time.

c, then immediately "re-flips" c, ready for its next use. And so its next use. And so its next use. And so its
"ready for its next use. And so its next use. A

How *should* simulation work? skip # *I^A* inlined. vaa din diaanvil v

$$
I_{\text{c}}\n v:= \text{HT} ; \n v:= \text{HT} ;\n g:= \text{Fil}_{\text{a}}(t) \n s:= \text{Fil}_{\text{c}}(t) \n F_{\text{a}} \n t_{\text{c}}
$$

$$
I_{c}
$$

$$
v := H \sqcap T;
$$

$$
g := F \text{Lip}(s)
$$

$$
F_{c}
$$

able c *cannot be accessed or observed* by the calling program, suces for

$$
I_A
$$

v := H Π
g := F Lip_A
 F_A

g:= H ¹*/*2 T; # Flip*A*.

 I_c

skip # *F^A* inlined.

IA

IC

v:= H u T; # Program. assumption into account.

 $v := H \sqcap v := T$ $T:=H\sqcap V:=T$

coin: Halland and Halland
Coine and Halland and Halland

But is this enough to prove refinement?

$$
I_{\alpha}
$$

\n
$$
v := H \cap T;
$$

\n
$$
g := F1ip_{\alpha}
$$

\n
$$
F_{\alpha}
$$

\n
$$
F_{\alpha}
$$

\n
$$
I_{\alpha}
$$

\n
$$
v := H \cap T;
$$

\n
$$
g := F1ip_{\alpha}
$$

\n
$$
F_{\alpha}
$$

\n
$$
I_{\alpha}
$$

\n
$$
v := H \cap T;
$$

\n
$$
g := F1ip_{\alpha}
$$

\n
$$
I_{\alpha}
$$

\n
$$
g := F1ip_{\alpha}
$$

\n
$$
I_{\alpha}
$$

\n
$$
f_{\alpha}
$$

able c *cannot be accessed or observed* by the calling program, suces for

IA: skip

1T;	I_c	Glue a series of title proofs g := Filup _C)	Glue a series of little proofs together via the rep...
-----	-------	---	---

v

Glue a series of little proofs together via the rep…

 $c := H \bigoplus_{1/2} c := T$

 $c := H \oplus_{1/2} c := T$

If this little proof is valid, it means that probability distributes with non-determinisim

 $v := H \sqcap v := T$

 $v := H \cap v := T$
 $c := H \oplus_{1/2} c := T$
 $c := H \cap v := T$
 $v := H \cap v := T$

But this in NOT valid using an MDP semantics for probability, which does not distinguish hidden statements, i.e. c

I A $v:= H \sqcap T;$ $F_{\rm A}$ and $F_{\rm C}$ $-F$ Γ Γ Γ Γ Γ Γ Γ Γ A $F_{\tt A}$

How *should* simulation work? v: Dinnendidon WOIIN. $\prod_{a \neq b} |A_{a} - I_{a}| \leq \frac{1}{2}$ **ing the Deference of the Deference to Def. 2, and the observable tensor to the observable tensor to the observable tensor that the observable tensor that the observable tensor** \mathbb{R} **.** skip # *I^A* inlined. vaa din dia dia 1911.

${}_{\rm Flip_4}$

v:= H u T; # Program. assumption into account.

But is this enough to prove refinement?

I_A	?	I_c	This is not a valid v:= H $\sqcap T$;	This is not a valid proof of g := Flip()	This is not a valid proof of g := Flip()	This is not a valid proof of refinement with hidden state and probability...
-------	---	-------	---	--	--	--

able c *cannot be accessed or observed* by the calling program, suces for

return coin

v

hidden state and probability…

?

Definition 3. *We say that an operation rep is a* simulation *from* (*I, OP, F*) *to* (I', OP', F') *if -using* $j \in J$ *to index corresponding operations in OP and OP'-* $\begin{array}{c} (1, 01, 1) \text{ if } -$ *the following inequations hold [12]*: **Definition 3.** We say that an operation rep is a simulation from (I, OP, F) to (I, OP, F) . $\frac{1}{2}$ $\frac{1}{2}$

I;*P*(*OP*); *F* v *I*⁰

;*P*(*OP* ⁰

); *F*⁰

,

What do we learn from this? There is a wealth of literature on verification methods for proving refinements on \sim v:= H u T; # Program. v:= H u T; # Program.

of datatypes (Def. 2). A common method is *simulation*:

andotorministic required, namely that is represented, namely that I $\frac{1}{\sqrt{N}}$ $\mathbf{v} := \mathbf{H} \sqcup \mathbf{T}$; with Parties with the common probability of the common probability $\frac{1}{D}$ Demonic nondeterministic choice cannot be resolved on the basis of internal state that it cannot "access" or "observe"

Figure 1 depicts two probabilistic datatypes. The left-hand datatype Fig. 1a has a

In order however for simulation (Def. 3) to establish refinement (Def. 3) to establish refinement (Def. 3) to e
In the simulation (Def. 3) to establish refinement (Def. 2), and (Def. 2), and (Def. 2), and (Def. 2), and (De

 $v:= H \sqcap T;$ $g :=$ Flip()

 F and F and F and F is matter. MDP's do not support a copy rule that distinguishes hidden (c state) from observed state.

I

What is the right semantics for probabilistic datatypes so that internal state is "invisible" to external nondeterminism?

Partially Observable Hidden Markov Models!

The abstract datatype has a visible coin to flip; the concrete datatype has a hidden c to flip

MDP

$coin := H_{1/2} \oplus coin := T$

$$
\mathcal{V} \longrightarrow \mathbb{P} \mathbb{D} \mathcal{V}
$$

MDP versus POMDP's

External nondeterminism cannot "observe" hidden coin flips inside the module

MDP

$\text{coin} := H_{1/2} \oplus \text{ coin} := T$

$$
\mathcal{V}\longrightarrow \mathbb{P} \mathbb{D} \mathcal{V}
$$

MDP versus POMDP's

External nondeterminism cannot "observe" hidden coin flips inside the module

External nondeterminism cannot "observe" hidden coin $coin := H$ $_{1/2} \oplus$ $coin := T$ $v := 0 \quad \Box \quad v := 1$ $c := H_{1/2} \oplus T$ $v := 0 \quad \Box \quad v := 1$

flips inside the module

MDP

POMDP

 $u[coin = T \land v = 0, coin = H \land v = 1]$ $u[coin = T \wedge v = 0, coin = H \wedge v = 0]$

MDP versus POMDP's $\nu \rightarrow \mathbb{P} \mathbb{D} \mathcal{V}$ $V \times \mathbb{D} \mathcal{H} \longrightarrow \mathbb{P} \mathbb{D}(\mathcal{V} \times \mathbb{D} \mathcal{H})$ $\left\{\n\begin{array}{l}\n\text{u}[\text{coin} = \text{T} \land \text{v} = \text{0}, \text{coin} = \text{H} \land \text{v} = \text{1} \\
\text{u}[\text{coin} = \text{T} \land \text{v} = \text{0}, \text{coin} = \text{H} \land \text{v} = \text{0}]\n\end{array}\n\right\}$ $u[coin = H \wedge v = 0, coin = T \wedge v = 1]$ $u[coin = T \wedge v = 1, coin = H \wedge v = 1]$ ${\rm v} = {\rm 0} \wedge {\rm u}[{\rm c} = {\rm T}, {\rm c} = {\rm H}]$

 $v = 1 \wedge u[c = T, c = H]$

Simulation now works in POMDP's, consistent with copy rule! v:= H u T; # Program. *i*² *Consistent with con* v:= H u T; # Program. $R_{12} = 2.5$ n now works in PUMDF's, consistent with copy rule assumption into account. $\sum_{i=1}^{n}$ **I***A I***A** *I***A IA IA IA IA IA** vin Tour B, combined

coin: Halland and Halland
Coine and Halland and Halland

$$
I_{A}
$$

\n
$$
v := H \sqcap T;
$$

\n
$$
g := F \text{Lip}_{A}
$$

\n
$$
F_{A}
$$

\n
$$
I_{C}
$$

\n
$$
v := H
$$

\n
$$
g := F
$$

\n
$$
F_{C}
$$

${}_{\rm Flip_{{\cal A}}}$

$$
I_{A}
$$
\n
$$
v := H \cap T;
$$
\n
$$
g := F \text{Lip}_{A}(x)
$$
\n
$$
F_{A}
$$
\n
$$
I_{C}
$$
\n
$$
v := H \cap T;
$$
\n
$$
g := F \text{Lip}_{C}(x)
$$
\n
$$
F_{C}
$$
\nThis little proof is
\n
$$
v := H \cap T;
$$
\n
$$
g := F \text{Lip}_{C}(x)
$$

return communications

c:= H ¹*/*2 T; # Pre-flip.

I

v

now valid!

What does this mean for these developers? ial does this ineall for these develop assumption into account.

able c *cannot be accessed or observed* by the calling program, suces for

var coin # Local variable

IA: skip Flip*A*: # Flip on demand. coin:= H $_{1/2} \oplus$ coin:= T; return coin *FA*: skip (a) The abstract datatype

ITHE SHOULD BE SCHOOL SHOULD BE SHOULD BE " Why should the concrete refine the abstract?

- Probabilistic invariants are sometimes simulation relations, and all its desired properties preserved, we say that (*I, OP, F*) is *itself* refined by *d I*⁰ *documents* are the refinement relation between programs, so \sim *O*² *document* relationships the refinement of \sim *O*² \sim *document* relationships the refinement of \sim *document* relationships the r
- Refinement depends on run-time information leaks concerning about the hidden state **combination of probability** \mathbf{r} and \mathbf{r} in the next \mathbf{r} is the ω setting is sequential programs defined by the by the basic program construction of ω sequential composition, assignment, branching and loops, and also both demonitories of the monitories of the monit ormation leaks concerning about ρ , ρ ,

corresponding operators in OP and OP', we have

section we review some known review some known results with an example. We review the construction with an exa
In the construction with an example. We review the construction with an example. We reconstruct the constructio

•

A small cadenza… $\frac{1}{2}$ of (desired) properties those programs should have: in its most general sense

refinement therefore means "preservation of desired properties". ³ In particular,

if *any* calling program replacing a datatype (*I, OP, F*) by (*I*⁰

, OP ⁰

*, F*⁰

 \mathscr{W}

 $I; \mathcal{P}(\mathit{OP}); F \subseteq I'; \mathcal{P}(\mathit{OP}'); F'$,

Definition 2. *A datatype* (I, OP, F) *is* refined by (I', OP', F') [12] if, for every *program P expressible using the constructs mentioned above, including calls on*

where " ;*" indicates sequential composition.*

How does this work for our challenge problem?

- unlike the $\{inv\}$ assertion style.
-

We can model things like secure implementation of cloud storage…

return "yes/no" without leaking any other information.

Conclusions for today

- What happens when some of the behaviour can be probabilistic in sequential programs? two program fragments (4) –right then down vs. down then right– are the same;
- ❖ Do the traditional proof methods for sequential programs (eg simulation) still work? α that α is our that the refinement distribution that the external through the external t ϵ traditional $\mathbf j$

 $((g, v), -) \longrightarrow \{((g, 0), -), ((g, 1), -)\}$

 $v:= 0 - 1$

c:= $0_{1/2} \oplus 1$ c:=0_{1/2} $\oplus 1$

 $((g, v), \mathsf{u}\{0, 1\})$ \longrightarrow \longrightarrow $\{((g, 0), \mathsf{u}\{0, 1\}), ((g, 1), \mathsf{u}\{0, 1\}) \}$ $v:= 0 \Box 1$

- ❖ Can we still use the abstract specification to prove properties of programs that use datatypes? Note that the *second* component of the state is a *distribution* (uniform u) over *H*, the e suil use the abstract specification to prove pro
- ❖ If they don't, what must be changed? $1 / 1 / 1$

 $V\times \mathbb{D}\mathcal{H} \rightarrow \mathbb{P}\mathbb{D}(\mathcal{V}\times \mathbb{D}\mathcal{H})$

- \triangleleft A semantics and refinement that distinguishes hidden and visible state; *existence of a simulation (Def. 3) which maps hidden state in in the abstract*
- \bullet We can now talk about information leaks.

Proof. (Sketch) We use the assumption here that rep distributes through all pro-

gram operators, including external demonic nondeterminism, of the calling pro-

Simulation

eveloped at ...

works!