

# Approximate Model Counting

Yong Kiam Tan and Jiong Yang

May 26, 2024

## Abstract

Approximate model counting is the task of approximating the number of solutions to an input formula. This entry formalizes `ApproxMC`, an algorithm due to Chakraborty et al. [1] with a probably approximately correct (PAC) guarantee, i.e., `ApproxMC` returns a multiplicative  $(1 + \varepsilon)$ -factor approximation of the model count with probability at least  $1 - \delta$ , where  $\varepsilon > 0$  and  $0 < \delta \leq 1$ . The algorithmic specification is further refined to a verified certificate checker that can be used to validate the results of untrusted `ApproxMC` implementations (assuming access to trusted randomness).

## Contents

<b>1</b>	<b>Preliminary probability/UHF lemmas</b>	<b>2</b>
<b>2</b>	<b>Random XORs</b>	<b>9</b>
2.1	Independence properties of random XORs . . . . .	13
2.2	Independence for repeated XORs . . . . .	37
<b>3</b>	<b>Random XOR hash family</b>	<b>44</b>
<b>4</b>	<b>ApproxMCCore definitions</b>	<b>51</b>
<b>5</b>	<b>ApproxMCCore analysis</b>	<b>63</b>
<b>6</b>	<b>ApproxMC definition and analysis</b>	<b>91</b>
<b>7</b>	<b>Certificate-based ApproxMC</b>	<b>107</b>
7.1	ApproxMC with lists instead of sets . . . . .	108
7.2	ApproxMC certificate checker . . . . .	118
<b>8</b>	<b>ApproxMC certification for CNF-XOR</b>	<b>132</b>
8.1	Blasting XOR constraints to CNF . . . . .	134
8.2	Export code for a SML implementation. . . . .	145

# 1 Preliminary probability/UHF lemmas

This section proves some simplified/specialized forms of lemmas that will be used in the algorithm's analysis later.

```
theory ApproxMCPreliminaries imports
  Frequency-Moments.Probability-Ext
  Concentration-Inequalities.Bienaymes-Identity
  Concentration-Inequalities.Paley-Zygmund-Inequality
begin
```

```
lemma card-inter-sum-indicat-real:
  assumes finite A
  shows  $\text{card } (A \cap B) = \text{sum } (\text{indicat-real } B) A$ 
  by (simp add: assms indicator-def)
```

```
lemma card-dom-ran:
  assumes finite D
  shows  $\text{card } \{w. \text{dom } w = D \wedge \text{ran } w \subseteq R\} = \text{card } R \wedge \text{card } D$ 
  using assms
```

```
proof (induct rule: finite-induct)
  case empty
  have  $\{w.: 'a \Rightarrow 'b \text{ option. } w = \text{Map.empty} \wedge \text{ran } w \subseteq R\} = \{\text{Map.empty}\}$ 
    by auto
  then show ?case
    by auto
```

```
next
  case (insert a A)
  have 1:  $\text{inj-on } (\lambda(w, r). w(a \mapsto r))$ 
    ( $\{w. \text{dom } w = A \wedge \text{ran } w \subseteq R\} \times R$ )
    unfolding inj-on-def
    by (smt (verit, del-insts) CollectD SigmaE fun-upd-None-if-notin-dom
local.insert(2) map-upd-eqD1 prod.simps(2) restrict-complement-singleton-eq
restrict-upd-same)
```

```
  have 21:  $(\lambda(w, r). w(a \mapsto r))$  ‘
    ( $\{w. \text{dom } w = A \wedge \text{ran } w \subseteq R\} \times R$ )  $\subseteq$ 
     $\{w. \text{dom } w = \text{insert } a A \wedge \text{ran } w \subseteq R\}$ 
    unfolding image-def
    using CollectD local.insert(2) by force
```

```
  have  $\bigwedge x. \text{dom } x = \text{insert } a A \implies$ 
     $\text{ran } x \subseteq R \implies$ 
     $\exists xa. \text{dom } xa = A \wedge$ 
     $\text{ran } xa \subseteq R \wedge (\exists y \in R. x = xa(a \mapsto y))$ 
    by (smt (verit, del-insts) Diff-insert-absorb domD dom-fun-upd
fun-upd-triv fun-upd-upd insert.hyps(2) insertCI insert-subset ran-map-upd)
```

```
  then have 22:
     $\{w. \text{dom } w = \text{insert } a A \wedge \text{ran } w \subseteq R\} \subseteq$ 
     $(\lambda(w, r). w(a \mapsto r))$  ‘
```

$(\{w. \text{dom } w = A \wedge \text{ran } w \subseteq R\} \times R)$   
**by** (*clarsimp simp add: image-def*)

**have** *bij-betw*  $(\lambda(w,r). w(a \mapsto r))$   
 $(\{w. \text{dom } w = A \wedge \text{ran } w \subseteq R\} \times R)$   
 $\{w. \text{dom } w = \text{insert } a \ A \wedge \text{ran } w \subseteq R\}$   
**unfolding** *bij-betw-def*  
**using** 1 21 22 **by** *clarsimp*

**then have**  $\text{card } \{w. \text{dom } w = \text{insert } a \ A \wedge \text{ran } w \subseteq R\} = \text{card } (\{w. \text{dom } w = A \wedge \text{ran } w \subseteq R\} \times R)$   
**by** (*auto simp add: bij-betw-same-card 1 21 22*)

**moreover have**  $\dots = \text{card } R \wedge \text{card } A * \text{card } R$   
**by**(*subst card-cartesian-product*) (*use insert.hyps(3) in auto*)

**ultimately show** *?case*  
**using** *insert.hyps* **by** (*auto simp add: card-insert-if*)  
**qed**

**lemma** *finite-set-pmf-expectation-sum*:  
**fixes**  $f :: 'a \Rightarrow 'c \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$   
**assumes** *finite* (*set-pmf A*)  
**shows** *measure-pmf.expectation A*  $(\lambda x. \text{sum } (f \ x) \ T) =$   
 $(\sum i \in T. \text{measure-pmf.expectation } A \ (\lambda x. f \ x \ i))$   
**apply** (*intro Bochner-Integration.integral-sum integrable-measure-pmf-finite*)  
**using** *assms* **by** *auto*

**lemma** (*in prob-space*) *k-universal-prob-unif*:  
**assumes** *k-universal k H D R*  
**assumes**  $w \in D \ \alpha \in R$   
**shows**  $\text{prob } \{s \in \text{space } M. H \ w \ s = \alpha\} = 1 / \text{card } R$   
**proof** –  
**have** *uniform-on*  $(H \ w) \ R$   
**using** *assms* **unfolding** *k-universal-def*  
**by** *auto*

**from** *uniform-onD[OF this]*  
**have** *prob*  
 $\{\omega \in \text{space } M. H \ w \ \omega \in \{\alpha\}\} =$   
 $\text{real } (\text{card } (R \cap \{\alpha\})) / \text{real } (\text{card } R) .$

**thus** *?thesis*  
**using** *assms* **by** *auto*  
**qed**

**lemma** *k-universal-expectation-eq*:  
**assumes**  $p: \text{finite } (\text{set-pmf } p)$

**assumes** *ind*: *prob-space.k-universal p k H D R*  
**assumes** *S*: *finite S S ⊆ D*  
**assumes** *a*:  $\alpha \in R$   
**shows**  
*prob-space.expectation p*  
 $(\lambda s. \text{real} (\text{card} (S \cap \{w. H w s = \alpha\}))) =$   
 $\text{real} (\text{card} S) / \text{card} R$   
**proof** –  
**have** *1*: *prob-space (measure-pmf p)*  
**by** (*simp add: prob-space-measure-pmf*)  
**have** *2*: *space (measure-pmf p) = UNIV* **by** *auto*  
**from** *prob-space.k-universal-prob-unif[OF 1 ind - a]*  
**have** *\**:  $\bigwedge w. w \in S \implies$   
*prob-space.prob p {s. H w s = α} = 1 / real (card R)*  
**using** *S(2)* **by** *auto*  
  
**have** *measure-pmf.expectation p*  
 $(\lambda s. \text{real} (\text{card} (S \cap \{w. H w s = \alpha\}))) =$   
*measure-pmf.expectation p*  
 $(\lambda s. \text{sum} (\text{indicat-real} \{w. H w s = \alpha\}) S)$   
**unfolding** *card-inter-sum-indicat-real[OF S(1)]*  
**by** *presburger*  
  
**moreover have** ... =  
 $(\sum_{i \in S.}$   
*measure-pmf.expectation p*  
 $(\text{indicat-real} \{s. H i s = \alpha\}))$   
**apply** (*subst finite-set-pmf-expectation-sum*)  
**using** *assms unfolding indicator-def* **by** *auto*  
  
**moreover have** ... =  
 $(\sum_{i \in S.}$   
*measure-pmf.prob p {s. H i s = α}*)  
**by** *auto*  
  
**moreover have** ... =  $(\sum_{i \in S.} 1 / \text{card} R)$   
**using** *\** **by** *auto*  
  
**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** (*in prob-space*) *two-universal-indep-var*:  
**assumes** *k-universal 2 H D R*  
**assumes**  $w \in D w' \in D w \neq w'$   
**shows** *indep-var*  
*borel*  
 $(\lambda x. \text{indicat-real} \{w. H w x = \alpha\} w)$   
*borel*  
 $(\lambda x. \text{indicat-real} \{w. H w x = \alpha\} w')$

**proof** –  
**have**  $Y: (\lambda z. (of\text{-}bool\ (z = \alpha))::real) \in (count\text{-}space\ UNIV) \rightarrow_M$   
*borel*  
**by** *auto*

**have** *k-wise-indep-vars 2*  
 $(\lambda\cdot. count\text{-}space\ UNIV)$   
 $H\ D$   
**using** *assms*  
**unfolding** *k-universal-def*  
**by** *auto*

**then have** *indep-vars*  $(\lambda\cdot. count\text{-}space\ UNIV)\ H\ \{w, w'\}$   
**unfolding** *k-wise-indep-vars-def*  
**by**  $(metis\ UNIV\text{-}bool\ assms(2)\ assms(3)\ card.empty\ card.insert\ card\text{-}UNIV\text{-}bool\ card\text{-}insert\text{-}le\ empty\text{-}iff\ empty\text{-}subsetI\ finite.emptyI\ finite.insertI\ insert\text{-}subset\ order.refl\ singletonD\ singleton\text{-}insert\text{-}inj\text{-}eq')$

**from** *indep-var-from-indep-vars* $[OF\ assms(4)\ this]$   
**have** *indep-var*  
 $(count\text{-}space\ UNIV)\ (H\ w)$   
 $(count\text{-}space\ UNIV)\ (H\ w') .$

**from** *indep-var-compose* $[OF\ this\ Y\ Y]$   
**show** *?thesis*  
**unfolding** *indicator-def*  
**by**  $(auto\ simp\ add: o-def)$

**qed**

**lemma** *two-universal-variance-bound:*

**assumes**  $p: finite\ (set\text{-}pmf\ p)$   
**assumes**  $ind: prob\text{-}space.k\text{-}universal\ (measure\text{-}pmf\ p)\ 2\ H\ D\ R$   
**assumes**  $S: finite\ S\ S \subseteq D$   
**assumes**  $a: \alpha \in R$

**shows**  
 $measure\text{-}pmf.variance\ p$   
 $(\lambda s. real\ (card\ (S \cap \{w. H\ w\ s = \alpha\}))) \leq$   
 $measure\text{-}pmf.expectation\ p$   
 $(\lambda s. real\ (card\ (S \cap \{w. H\ w\ s = \alpha\})))$

**proof** –  
**have**  $vb: measure\text{-}pmf.variance\ p$   
 $(\lambda x. (indicat\text{-}real\ \{w. H\ w\ x = \alpha\}\ i)) \leq$   
 $measure\text{-}pmf.expectation\ p$   
 $(\lambda x. (indicat\text{-}real\ \{w. H\ w\ x = \alpha\}\ i))$  **for**  $i$

**proof** –  
**have**  $measure\text{-}pmf.variance\ p$   
 $(\lambda x. (indicat\text{-}real\ \{w. H\ w\ x = \alpha\}\ i)) =$   
 $measure\text{-}pmf.expectation\ p$

$(\lambda x. (\text{indicat-real } \{w. H w x = \alpha\} i)^2) -$   
 $(\text{measure-pmf.expectation } p$   
 $(\lambda x. \text{indicat-real } \{w. H w x = \alpha\} i))^2$   
**apply** (*subst measure-pmf.variance-eq*)  
**by** (*auto simp add: p integrable-measure-pmf-finite*)  
**moreover have**  $\dots \leq \text{measure-pmf.expectation } p$   
 $(\lambda x. (\text{indicat-real } \{w. H w x = \alpha\} i)^2)$   
**by** *simp*  
**moreover have**  $\dots = \text{measure-pmf.expectation } p$   
 $(\lambda x. (\text{indicat-real } \{w. H w x = \alpha\} i))$   
**by** (*metis (mono-tags, lifting) indicator-simps(1) indicator-simps(2)*  
*power2-eq-1-iff zero-eq-power2*)  
**ultimately show** *?thesis* **by** *linarith*  
**qed**

**have** *measure-pmf.variance*  $p$   
 $(\lambda s. \text{real } (\text{card } (S \cap \{w. H w s = \alpha\}))) =$   
 $\text{measure-pmf.variance } p$   
 $(\lambda s. \text{sum } (\text{indicat-real } \{w. H w s = \alpha\}) S)$   
**unfolding** *card-inter-sum-indicat-real[OF S(1)]*  
**by** *presburger*

**then have**  $\dots = (\sum_{i \in S.}$   
 $\text{measure-pmf.variance } p$   
 $(\lambda x. (\text{indicat-real } \{w. H w x = \alpha\} i)))$   
**apply** (*subst measure-pmf.bienaymes-identity-pairwise-indep*)  
**using** *S prob-space-measure-pmf*  
**by** (*auto intro!: prob-space.two-universal-indep-var[OF - ind] simp*  
*add: p integrable-measure-pmf-finite*)

**moreover have**  $\dots \leq (\sum_{i \in S.}$   
 $\text{measure-pmf.expectation } p$   
 $(\lambda x. (\text{indicat-real } \{w. H w x = \alpha\} i)))$   
**by** (*simp add: sum-mono vb*)  
**moreover have**  $\dots = \text{measure-pmf.expectation } p$   
 $(\lambda s. \text{sum } (\text{indicat-real } \{w. H w s = \alpha\}) S)$   
**apply** (*subst finite-set-pmf-expectation-sum*)  
**using** *assms* **by** *auto*  
**ultimately show** *?thesis*  
**by** (*simp add: assms(3) card-inter-sum-indicat-real*)  
**qed**

**lemma** (*in prob-space*) *k-universal-mono*:  
**assumes**  $k' \leq k$   
**assumes** *k-universal k H D R*  
**shows** *k-universal k' H D R*  
**using** *assms*  
**unfolding** *k-universal-def k-wise-indep-vars-def*  
**by** *auto*

**lemma** *finite-set-pmf-expectation-add*:  
**assumes** *finite* (*set-pmf* *S*)  
**shows**  $\text{measure-pmf.expectation } S (\lambda x. ((f x)::\text{real}) + g x) =$   
 $\text{measure-pmf.expectation } S f + \text{measure-pmf.expectation } S g$   
**by** (*auto intro: Bochner-Integration.integral-add simp add: assms integrable-measure-pmf-finite*)

**lemma** *finite-set-pmf-expectation-add-const*:  
**assumes** *finite* (*set-pmf* *S*)  
**shows**  $\text{measure-pmf.expectation } S (\lambda x. ((f x)::\text{real}) + g) =$   
 $\text{measure-pmf.expectation } S f + g$

**proof** –  
**have**  $g = \text{measure-pmf.expectation } S (\lambda x. g)$   
**by** *simp*  
**thus** *?thesis*  
**by** (*simp add: assms finite-set-pmf-expectation-add*)  
**qed**

**lemma** *finite-set-pmf-expectation-diff*:  
**assumes** *finite* (*set-pmf* *S*)  
**shows**  $\text{measure-pmf.expectation } S (\lambda x. ((f x)::\text{real}) - g x) =$   
 $\text{measure-pmf.expectation } S f - \text{measure-pmf.expectation } S g$   
**by** (*auto intro: Bochner-Integration.integral-diff simp add: assms integrable-measure-pmf-finite*)

**lemma** *spec-paley-zygmund-inequality*:  
**assumes** *fin*: *finite* (*set-pmf* *p*)  
**assumes** *Zpos*:  $\bigwedge z. Z z \geq 0$   
**assumes** *t*:  $\vartheta \leq 1$   
**shows**  
 $(\text{measure-pmf.variance } p Z + (1-\vartheta)^2 * (\text{measure-pmf.expectation } p Z)^2) *$   
 $\text{measure-pmf.prob } p \{z. Z z > \vartheta * \text{measure-pmf.expectation } p Z\}$   
 $\geq$   
 $(1-\vartheta)^2 * (\text{measure-pmf.expectation } p Z)^2$

**proof** –  
**have** *prob-space* (*measure-pmf* *p*) **by** (*auto simp add: prob-space-measure-pmf*)

**from** *prob-space.paley-zygmund-inequality*[*OF this - integrable-measure-pmf-finite*[*OF fin*] *t*]

**show** *?thesis*  
**using** *Zpos* **by** *auto*  
**qed**

**lemma** *spec-chebyshev-inequality*:  
**assumes** *fin*: *finite* (*set-pmf* *p*)  
**assumes** *pvar*:  $\text{measure-pmf.variance } p Y > 0$

**assumes**  $k: k > 0$   
**shows**  
 $\{y. (Y y - \text{measure-pmf.expectation } p Y)^2 \leq k^2 * \text{measure-pmf.variance } p Y\} \leq 1 / k^2$   
**proof** –  
**define**  $f$  **where**  
 $f x = Y x / \text{sqrt}(\text{measure-pmf.variance } p Y)$  **for**  $x$   
**have**  $1$ :  
 $\text{measure-pmf.random-variable } p \text{ borel } f$   
**by** *auto*  
**have**  $*$ :  $(\lambda x. (f x)^2) = (\lambda x. (Y x)^2 / \text{measure-pmf.variance } p Y)$   
**unfolding**  $f\text{-def}$   
**by** (*simp add: power-divide*)  
**have**  $2$ :  
 $\text{integrable } p (\lambda x. (f x)^2)$   
**unfolding**  $*$   
**by** (*intro integrable-measure-pmf-finite[OF fin]*)  
**from**  
 $\text{measure-pmf.Chebyshev-inequality[OF 1 2 k]}$   
**have**  $\text{ineq1:measure-pmf.prob } p$   
 $\{x. k \leq |f x - \text{measure-pmf.expectation } p f|\}$   
 $\leq \text{measure-pmf.expectation } p$   
 $(\lambda x. (f x - \text{measure-pmf.expectation } p f)^2) / k^2$  **by** *auto*  
  
**have**  $(\lambda x. (f x - \text{measure-pmf.expectation } p f)^2) =$   
 $(\lambda x. ((Y x - \text{measure-pmf.expectation } p Y) / \text{sqrt}(\text{measure-pmf.variance } p Y))^2)$   
**unfolding**  $f\text{-def}$   
**by** (*simp add: diff-divide-distrib*)  
  
**moreover have**  $\dots = (\lambda x. (Y x - \text{measure-pmf.expectation } p Y)^2$   
 $/ (\text{sqrt}(\text{measure-pmf.variance } p Y))^2)$   
**by** (*simp add: power-divide*)  
**moreover have**  $\dots = (\lambda x. (Y x - \text{measure-pmf.expectation } p Y)^2$   
 $/ \text{measure-pmf.variance } p Y)$   
**by** *simp*  
**ultimately have**  $\text{unfold}:(\lambda x. (f x - \text{measure-pmf.expectation } p f)^2)$   
 $= (\lambda x. (Y x - \text{measure-pmf.expectation } p Y)^2 /$   
 $\text{measure-pmf.variance } p Y)$   
**by** *auto*  
**then have**  $\text{measure-pmf.expectation } p (\lambda x. (f x - \text{measure-pmf.expectation } p f)^2) / k^2$   
 $= \text{measure-pmf.expectation } p (\lambda x. (Y x - \text{measure-pmf.expectation } p Y)^2 / \text{measure-pmf.variance } p Y) / k^2$   
**by** *auto*  
**moreover have**  $\dots = \text{measure-pmf.variance } p Y / \text{measure-pmf.variance } p Y / k^2$   
**by** *simp*



```

moreover have ... = 1 / k2
  using pvar by force
  ultimately have eq1:measure-pmf.expectation p (λx. (f x - mea-
    sure-pmf.expectation p f)2) / k2 = 1 / k2
    by auto
    have (λx. k ≤ |f x - measure-pmf.expectation p f|) = (λx. k2 ≤ (f
      x - measure-pmf.expectation p f)2)
      by (metis (no-types, opaque-lifting) abs-of-nonneg k less-le real-le-rsqrt
        real-sqrt-abs sqrt-ge-absD)
    moreover have ... = (λx. k2 ≤ ((Y x - measure-pmf.expectation p
      Y)2 / measure-pmf.variance p Y))
      by (metis unfold)
    moreover have ... = (λx. (Y x - measure-pmf.expectation p Y)2
      ≥ k2 * measure-pmf.variance p Y)
      by (simp add: pos-le-divide-eq pvar)
    ultimately have cond:(λx. k ≤ |f x - measure-pmf.expectation p
      f|)
      = (λx. (Y x - measure-pmf.expectation p Y)2 ≥ k2 *
        measure-pmf.variance p Y)
      by auto
    show ?thesis using ineq1 cond eq1
      by auto
qed

end

```

## 2 Random XORs

The goal of this section is to prove that, for a randomly sampled XOR  $X$  from a set of variables  $V$ :

1. the probability of an assignment  $w$  satisfying  $X$  is  $\frac{1}{2}$ ;
2. for any distinct assignments  $w, w'$  the probability of both satisfying  $X$  is equal to  $\frac{1}{4}$  (2-wise independence); and
3. for any distinct assignments  $w, w', w''$  the probability of all three satisfying  $X$  is equal to  $\frac{1}{8}$  (3-wise independence).

**theory** *RandomXOR* **imports**

*ApproxMCPreliminaries*

*Universal-Hash-Families.Universal-Hash-Families-More-Product-PMF*

*Monad-Normalisation.Monad-Normalisation*

**begin**

A random XOR constraint is modeled as a random subset of variables and a randomly chosen RHS bit.

**definition** *random-xor* :: 'a set ⇒ ('a set × bool) pmf

**where** *random-xor* V =

*pair-pmf (pmf-of-set (Pow V)) (bernoulli-pmf (1/2))*

**lemma** *pmf-of-set-Pow-fin-map:*

**assumes** *V:finite V*

**shows** *pmf-of-set (Pow V) =*

*map-pmf (λb. {x ∈ V. b x = Some True})*

*(Pi-pmf V def (λ-. map-pmf Some (bernoulli-pmf (1 / 2))))*

**proof** –

**have** \*: *Pi-pmf V def (λ-. map-pmf Some (bernoulli-pmf (1 / 2)))*

=

*map-pmf (λf x. if x ∈ V then f x else def)*

*(Pi-pmf V (Some False) (λ-. map-pmf Some (bernoulli-pmf (1 / 2))))*

**unfolding** *Pi-pmf-default-swap[OF V] by auto*

**have** \*\*: *pmf-of-set (Pow V) =*

*map-pmf*

*(λx. {xa. (xa ∈ V → x xa) ∧ xa ∈ V})*

*(Pi-pmf V False (λ-. bernoulli-pmf (1 / 2)))*

**by** (*smt (verit, ccfv-SIG) Collect-cong pmf-of-set-Pow-conv-bernoulli V pmf.map-cong*)

**show** *?thesis*

**unfolding** \*

**apply** (*subst Pi-pmf-map[OF V, of - False]*)

**using** \*\* **by** (*auto simp add: pmf.map-comp o-def*)

**qed**

**lemma** *random-xor-from-bits:*

**assumes** *V:finite V*

**shows** *random-xor V =*

*pair-pmf*

*(map-pmf (λb. {x ∈ V. b x = Some True})*

*(Pi-pmf V def (λ-. map-pmf Some (bernoulli-pmf (1/2))))*

*(bernoulli-pmf (1/2))*

**unfolding** *random-xor-def*

**using** *V pmf-of-set-Pow-fin-map by fastforce*

**fun** *satisfies-xor* :: (*'a set × bool*) ⇒ *'a set* ⇒ *bool*

**where** *satisfies-xor (x,b) ω =*

*even (card (ω ∩ x) + of-bool b)*

**lemma** *satisfies-xor-inter:*

**shows** *satisfies-xor (ω ∩ x ,b) ω = satisfies-xor (x,b) ω*

**by** (*auto simp add: Int-commute*)

**lemma** *prob-bernoulli-bind-pmf*:  
**assumes**  $0 \leq p \leq 1$   
**assumes** *finite E*  
**shows** *measure-pmf.prob*  
 $(\text{bernoulli-pmf } p \gg x) E =$   
 $p * (\text{measure-pmf.prob } (x \text{ True}) E) +$   
 $(1 - p) * (\text{measure-pmf.prob } (x \text{ False}) E)$   
**using** *assms*  
**by** (*auto simp add: pmf-bind measure-measure-pmf-finite[OF assms(3)]*  
*vector-space-over-itself.scale-sum-right comm-monoid-add-class.sum.distrib*  
*mult commute*)

**lemma** *set-pmf-random-xor*:  
**assumes** *V: finite V*  
**shows** *set-pmf (random-xor V) = (Pow V)  $\times$  UNIV*  
**unfolding** *random-xor-def*  
**using** *assms by (auto simp add: Pow-not-empty Set.basic-monos(7))*

**lemma** *pmf-of-set-prod*:  
**assumes**  $P \neq \{\}$   $Q \neq \{\}$   
**assumes** *finite P finite Q*  
**shows** *pmf-of-set (P  $\times$  Q) = pair-pmf (pmf-of-set P) (pmf-of-set Q)*  
**by** (*auto intro!: pmf-eqI simp add: indicator-def pmf-pair assms*)

**lemma** *random-xor-pmf-of-set*:  
**assumes** *V: finite V*  
**shows** *random-xor V = pmf-of-set ((Pow V)  $\times$  UNIV)*  
**unfolding** *random-xor-def*  
**apply** (*subst pmf-of-set-prod*)  
**using** *V bernoulli-pmf-half-conv-pmf-of-set by auto*

**lemma** *prob-random-xor-with-set-pmf*:  
**assumes** *V: finite V*  
**shows** *prob-space.prob (random-xor V) {c. P c} =*  
*prob-space.prob (random-xor V) {c. fst c  $\subseteq$  V  $\wedge$  P c}*  
**by** (*smt (verit, best) PowD assms measure-pmf.measure-pmf-eq mem-Collect-eq*  
*mem-Sigma-iff prod.collapse set-pmf-random-xor*)

**lemma** *prob-set-parity*:  
**assumes** *measure-pmf.prob M*  
 $\{c. P c\} = q$   
**shows** *measure-pmf.prob M*  
 $\{c. P c = b\} = (\text{if } b \text{ then } q \text{ else } 1 - q)$   
**proof** –  
{  
**assume** *b:b*  
**have** *?thesis using assms*  
**using** *b by presburger*

```

}
moreover {
  assume  $b:\neg b$ 
  have  $\{c. P\ c\} \in \text{measure-pmf.events } M$  by auto
  from measure-pmf.prob-compl[OF this]
  have  $1 - q = \text{measure-pmf.prob } M (UNIV - \{c. P\ c\})$ 
    using assms by auto
  moreover have ... =
    prob-space.prob  $M$ 
     $\{c. P\ c = b\}$ 
    by (simp add: b measure-pmf.measure-pmf-eq)
  ultimately have ?thesis
    using  $b$  by presburger
}
ultimately show ?thesis by auto
qed

lemma satisfies-random-xor:
  assumes  $V: \text{finite } V$ 
  shows prob-space.prob (random-xor  $V$ )
     $\{c. \text{satisfies-xor } c\ \omega\} = 1 / 2$ 
proof -
  have eq:  $\{(c::'a\ \text{set} \times \text{bool}). \text{fst } c \subseteq V\} = \{c. c \subseteq V\} \times UNIV$ 
    by auto
  then have finite  $\{(c::'a\ \text{set} \times \text{bool}). \text{fst } c \subseteq V\}$ 
    using assms by auto

  then have *:
     $x \subseteq V \implies$ 
    measure-pmf.prob (bernoulli-pmf  $(1 / 2) \gg (\lambda y. \text{return-pmf } (x,$ 
     $y)))$ 
     $\{c. \text{fst } c \subseteq V \wedge \text{satisfies-xor } c\ \omega\} = 1 / 2$  for  $x$ 
    apply (subst prob-bernoulli-bind-pmf)
    by (auto simp add: indicator-def)

  have prob-space.prob (random-xor  $V$ )  $\{c. \text{satisfies-xor } c\ \omega\} =$ 
    prob-space.prob (random-xor  $V$ )  $\{c. \text{fst } c \subseteq V \wedge \text{satisfies-xor } c\ \omega\}$ 
    using prob-random-xor-with-set-pmf[OF V] by auto
  also have ... =
    prob-space.expectation (random-xor  $V$ )
    (indicat-real  $\{c. \text{fst } c \subseteq V \wedge \text{satisfies-xor } c\ \omega\}$ )
    by auto
  also have ... =  $(\sum a \in \text{Pow } V.$ 
    inverse (real (card (Pow  $V$ ))) *
    measure-pmf.prob
    (bernoulli-pmf  $(1 / 2) \gg$ 
     $(\lambda y. \text{return-pmf } (a, y)))$ 
     $\{c. \text{fst } c \subseteq V \wedge \text{satisfies-xor } c\ \omega\}$ )
    unfolding random-xor-def pair-pmf-def

```

```

apply (subst pmf-expectation-bind-pmf-of-set)
using assms by auto
also have ... = ( $\sum a \in \text{Pow } V.$ 
  inverse (real (card (Pow V))) * (1/2))
by (simp add: *)
also have ... =
  inverse (real (card (Pow V))) * (1/2) * ( $\sum a \in \text{Pow } V. 1$ )
using * by force
also have ... = 1/2
by (metis One-nat-def Suc-leI assms card-Pow divide-real-def in-
verse-inverse-eq inverse-nonzero-iff-nonzero nat-one-le-power nonzero-mult-div-cancel-left
not-one-le-zero of-nat-eq-0-iff pos2 real-of-card)
finally show ?thesis
by presburger
qed

```

```

lemma satisfies-random-xor-parity:
assumes V: finite V
shows prob-space.prob (random-xor V)
  {c. satisfies-xor c  $\omega = b$ } = 1 / 2
using prob-set-parity[OF satisfies-random-xor[OF V]]
by auto

```

## 2.1 Independence properties of random XORs

```

lemma pmf-of-set-powerset-split:
assumes S  $\subseteq$  V finite V
shows
  map-pmf ( $\lambda(x,y). x \cup y$ )
  (pmf-of-set (Pow S  $\times$  Pow (V - S))) =
  pmf-of-set (Pow V)
proof -
have spmfs: set-pmf (pmf-of-set (Pow S)) = Pow S
using assms
by (auto intro!: set-pmf-of-set simp add: rev-finite-subset)
have spmfs: set-pmf (pmf-of-set (Pow (V-S))) = Pow (V-S)
using assms by (auto intro!: set-pmf-of-set)

have xsub: x  $\subseteq$  V  $\implies$ 
   $\exists xa \in \text{Pow } S.$ 
   $\exists y \in \text{Pow } (V - S). x = xa \cup y$  for x
by (metis Diff-subset-conv Pow-iff Un-Diff-Int basic-trans-rules(23)
inf-le2 sup-commute)

have inj: inj-on ( $\lambda(x, y). x \cup y$ )
  (Pow S  $\times$  Pow (V - S))
unfolding inj-on-def
by auto
then have bij: bij-betw ( $\lambda(x, y). x \cup y$ )

```

```

      ((Pow S) × Pow (V - S))
      (Pow V)
unfolding bij-betw-def
using assms(1) xsub by (auto simp add: image-def)

have map-pmf (λ(x, y). x ∪ y)
      (pmf-of-set (Pow S × Pow (V - S))) =
      pmf-of-set (Pow V)
apply (subst map-pmf-of-set-inj[OF inj])
subgoal by (auto simp add: image-def)
subgoal using bij assms(2) bij-betw-finite by blast
apply (intro arg-cong[where f = pmf-of-set])
using assms(1) xsub by (auto simp add: image-def)

thus ?thesis
unfolding spmfS spmfVS
by auto
qed

lemma pmf-of-set-Pow-sing:
shows pmf-of-set (Pow {x}) =
      bernoulli-pmf (1 / 2) ≫
      (λb. return-pmf (if b then {x} else {}))
apply (intro pmf-eqI)
apply (subst pmf-of-set)
by (auto simp add: pmf-bind card-Pow indicator-def subset-singleton-iff)

lemma pmf-of-set-sing-coin-flip:
assumes finite V
shows pmf-of-set (Pow {x} × Pow V) =
      map-pmf (λ(r,c). (if c then {x} else {}, r)) (random-xor V)
proof -
have *: pmf-of-set (Pow {x} × Pow V) =
      pair-pmf (pmf-of-set (Pow {x})) (pmf-of-set (Pow V))
apply (intro pmf-of-set-prod)
using assms by auto
show ?thesis
unfolding *
apply (intro pmf-eqI)
including monad-normalisation
by (auto simp add: map-pmf-def pair-pmf-def random-xor-def pmf-of-set-Pow-sing)
qed

lemma measure-pmf-prob-dependent-product-bound-eq:
assumes countable A ∧ i. countable (B i)
assumes ∧ a. a ∈ A ⇒ measure-pmf.prob N (B a) = r
shows measure-pmf.prob (pair-pmf M N) (Sigma A B) =
      measure-pmf.prob M A * r

```

**proof** –  
**have**  $\text{measure-pmf.prob } (\text{pair-pmf } M \ N) \ (\text{Sigma } A \ B) =$   
 $(\sum_a (a, b) \in \text{Sigma } A \ B. \text{ pmf } M \ a * \text{ pmf } N \ b)$   
**by** (*auto intro!: infsetsum-cong simp add: measure-pmf-conv-infsetsum pmf-pair*)  
**also have**  $\dots = (\sum_a a \in A. \sum_b b \in B \ a. \text{ pmf } M \ a * \text{ pmf } N \ b)$   
**apply** (*subst infsetsum-Sigma[OF assms(1-2)]*)  
**subgoal by** (*metis (no-types, lifting) SigmaE abs-summable-on-cong case-prod-conv pmf-abs-summable pmf-pair*)  
**by** (*auto simp add: assms case-prod-unfold*)  
  
**also have**  $\dots = (\sum_a a \in A. \text{ pmf } M \ a * (\text{measure-pmf.prob } N \ (B \ a)))$   
**by** (*simp add: infsetsum-cmult-right measure-pmf-conv-infsetsum pmf-abs-summable*)  
**also have**  $\dots = (\sum_a a \in A. \text{ pmf } M \ a * r)$   
**using**  $\text{assms}(3)$  **by force**  
**also have**  $\dots = \text{measure-pmf.prob } M \ A * r$   
**by** (*simp add: infsetsum-cmult-left pmf-abs-summable measure-pmf-conv-infsetsum*)  
**finally show** *?thesis*  
**by** *linarith*  
**qed**

**lemma** *measure-pmf-prob-dependent-product-bound-eq'*:  
**assumes**  $\text{countable } (A \cap \text{set-pmf } M) \wedge i. \text{countable } (B \ i \cap \text{set-pmf } N)$   
**assumes**  $\bigwedge a. a \in A \cap \text{set-pmf } M \implies \text{measure-pmf.prob } N \ (B \ a \cap \text{set-pmf } N) = r$   
**shows**  $\text{measure-pmf.prob } (\text{pair-pmf } M \ N) \ (\text{Sigma } A \ B) = \text{measure-pmf.prob } M \ A * r$   
**proof** –  
**have**  $*$ :  $\text{Sigma } A \ B \cap (\text{set-pmf } M \times \text{set-pmf } N) =$   
 $\text{Sigma } (A \cap \text{set-pmf } M) \ (\lambda i. B \ i \cap \text{set-pmf } N)$   
**by** *auto*  
  
**have**  $\text{measure-pmf.prob } (\text{pair-pmf } M \ N) \ (\text{Sigma } A \ B) =$   
 $\text{measure-pmf.prob } (\text{pair-pmf } M \ N) \ (\text{Sigma } (A \cap \text{set-pmf } M) \ (\lambda i. B \ i \cap \text{set-pmf } N))$   
**by** (*metis \* measure-Int-set-pmf set-pair-pmf*)  
**moreover have**  $\dots =$   
 $\text{measure-pmf.prob } M \ (A \cap \text{set-pmf } M) * r$   
**using** *measure-pmf-prob-dependent-product-bound-eq[OF assms(1-3)]*  
**by** *auto*  
**moreover have**  $\dots = \text{measure-pmf.prob } M \ A * r$   
**by** (*simp add: measure-Int-set-pmf*)  
**ultimately show** *?thesis* **by** *linarith*  
**qed**

**lemma** *single-var-parity-coin-flip*:

**assumes**  $x \in \omega$  *finite*  $\omega$   
**assumes** *finite*  $a$   $x \notin a$   
**shows**  $\text{measure-pmf.prob} (\text{pmf-of-set} (\text{Pow } \{x\}))$   
 $\{y. \text{even} (\text{card} ((a \cup y) \cap \omega)) = b\} = 1/2$   
**proof** –  
**have**  $\text{insert } x \ a \cap \omega = \text{insert } x \ (a \cap \omega)$   
**using** *assms* **by** *auto*  
**then have**  $*$ :  $\text{card} (\text{insert } x \ a \cap \omega) = 1 + \text{card} (a \cap \omega)$   
**by** (*simp* *add: assms(2) assms(4)*)  
  
**have**  $\text{measure-pmf.prob} (\text{pmf-of-set} (\text{Pow } \{x\}))$   
 $\{y. \text{even} (\text{card} ((a \cup y) \cap \omega)) = b\} =$   
 $\text{measure-pmf.prob} (\text{bernoulli-pmf} (1/2))$   
 $\{\text{odd} (\text{card} (a \cap \omega)) = b\}$   
**unfolding** *pmf-of-set-Pow-sing map-pmf-def[symmetric]*  
**by** (*auto intro!: measure-prob-cong-0 simp add:image-def \**)  
**moreover have**  $\dots = 1/2$   
**by** (*simp add: measure-pmf-single*)  
**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** *prob-pmf-of-set-nonempty-parity:*

**assumes**  $V$ : *finite*  $V$   
**assumes**  $x \in \omega$   $\omega \subseteq V$   
**assumes**  $\bigwedge c. c \in E \longleftrightarrow c - \{x\} \in E$   
**shows**  $\text{prob-space.prob} (\text{pmf-of-set} (\text{Pow } V))$   
 $(E \cap \{c. \text{even} (\text{card} (c \cap \omega)) = b\}) =$   
 $1 / 2 * \text{prob-space.prob} (\text{pmf-of-set} (\text{Pow} (V - \{x\}))) E$   
**proof** –  
**have**  $1$ :  $\text{set-pmf} (\text{pmf-of-set} (\text{Pow } \{x\})) = \text{Pow } \{x\}$   
**by** (*simp add: Pow-not-empty*)  
**have**  $2$ :  $\text{set-pmf} (\text{pmf-of-set} (\text{Pow} (V - \{x\}))) = \text{Pow} (V - \{x\})$   
**by** (*simp add: Pow-not-empty assms(1)*)  
**have**  $3$ :  $\text{set-pmf} (\text{pmf-of-set} (\text{Pow } \{x\} \times \text{Pow} (V - \{x\}))) = \text{Pow}$   
 $\{x\} \times \text{Pow} (V - \{x\})$   
**by** (*simp add: Pow-not-empty assms(1)*)  
  
**have**  $\{x\} \subseteq V$  **using** *assms* **by** *auto*  
**from** *pmf-of-set-powerset-split[OF this assms(1)]*  
**have**  $e$ :  $\text{map-pmf} (\lambda(x, y). x \cup y)$   
 $(\text{pmf-of-set} (\text{Pow } \{x\} \times \text{Pow} (V - \{x\}))) =$   
 $\text{pmf-of-set} (\text{Pow } V)$  **using**  $1\ 2$  **by** *auto*  
**have**  $\text{map-pmf} (\lambda(x, y). x \cup y)$   
 $(\text{pair-pmf} (\text{pmf-of-set} (\text{Pow } \{x\}))$   
 $(\text{pmf-of-set} (\text{Pow} (V - \{x\})))) =$   
 $\text{map-pmf} (\lambda(x, y). x \cup y)$   
 $(\text{pair-pmf} (\text{pmf-of-set} (\text{Pow} (V - \{x\})))$   
 $(\text{pmf-of-set} (\text{Pow } \{x\})))$



**apply** (*subst pair-commute-pmf*)  
**by** (*auto simp add: pmf.map-comp o-def case-prod-unfold Un-commute*)  
**then have** \*: *pmf-of-set (Pow V) =*  
*map-pmf (λ(x, y). x ∪ y)*  
*(pair-pmf (pmf-of-set (Pow (V - {x}))) (pmf-of-set (Pow {x})))*  
**unfolding** *e[symmetric]*  
**apply** (*subst pmf-of-set-prod*)  
**using** *V by auto*

**have** \*\*:  $((\lambda(x, y). x \cup y) -' (E \cap S)) \cap (Pow (V - \{x\}) \times Pow\{x\})$   
 $=$   
 $Sigma E (\lambda x. \{y. (x \cup y) \in S\}) \cap (Pow (V - \{x\}) \times Pow\{x\})$

**for** *S*

**proof** -

**have** 11:  $\bigwedge a b. a \cup b \in E \implies$

$a \cup b \in S \implies$

$a \subseteq V - \{x\} \implies$

$b \subseteq \{x\} \implies a \in E$

**by** (*metis Diff-insert-absorb Un-insert-right assms(4) boolean-algebra-cancel.sup0 subset-Diff-insert subset-singleton-iff*)

**have** 21:  $\bigwedge a b. a \in E \implies$

$a \cup b \in S \implies$

$a \subseteq V - \{x\} \implies$

$b \subseteq \{x\} \implies a \cup b \in E$

**by** (*metis Diff-cancel Un-Diff Un-empty-left assms(4) inf-sup-aci(5) subset-singletonD*)

**have** 1:

$\bigwedge ab. ab \in ((\lambda(x, y). x \cup y) -' (E \cap S)) \cap (Pow (V - \{x\}) \times Pow\{x\}) \implies$

$ab \in Sigma E (\lambda x. \{y. (x \cup y) \in S\}) \cap (Pow (V - \{x\}) \times Pow\{x\})$

**using** 11 **by** *clarsimp*

**also have** 2:

$\bigwedge ab. ab \in Sigma E (\lambda x. \{y. (x \cup y) \in S\}) \cap (Pow (V - \{x\}) \times Pow\{x\}) \implies$

$ab \in ((\lambda(x, y). x \cup y) -' (E \cap S)) \cap (Pow (V - \{x\}) \times Pow\{x\})$

**using** 21 **by** *clarsimp*

**ultimately show** *?thesis*

**apply** (*intro antisym*)

**by** (*meson subsetI*)+

**qed**

**have** *eR*:  $\bigwedge a. a \in E \cap set-pmf (pmf-of-set (Pow (V - \{x\}))) \implies$   
*measure-pmf.prob (pmf-of-set (Pow {x}))*  
 $(\{y. even (card ((a \cup y) \cap \omega)) = b\} \cap$

```

    set-pmf (pmf-of-set (Pow {x})) = 1 / 2
  apply (subst measure-Int-set-pmf)
  apply (intro single-var-parity-coin-flip)
  subgoal using assms by clarsimp
  subgoal using assms rev-finite-subset by blast
  subgoal by (metis 2 IntD2 PowD assms(1) finite-Diff rev-finite-subset)
  using 2 by blast

have
  prob-space.prob (pmf-of-set (Pow V))
    (E ∩ {c. even (card (c ∩ ω)) = b}) =
  prob-space.prob (map-pmf (λ(x, y). x ∪ y)
    (pair-pmf (pmf-of-set (Pow (V - {x}))) (pmf-of-set (Pow {x}))))
    (E ∩ {c. even (card (c ∩ ω)) = b}) unfolding * by auto
  moreover have ... =
    prob-space.prob (pair-pmf (pmf-of-set (Pow (V - {x}))) (pmf-of-set
      (Pow {x})))
      ((λ(x, y). x ∪ y) -' (E ∩ {c. even (card (c ∩ ω)) = b}))
    by auto
  moreover have ... =
    prob-space.prob (pair-pmf (pmf-of-set (Pow (V - {x}))) (pmf-of-set
      (Pow {x})))
      ((λ(x, y). x ∪ y) -' (E ∩ {c. even (card (c ∩ ω)) = b}) ∩ (Pow
        (V - {x}) × Pow{x}))
    by (smt (verit) 1 2 Int-iff Sigma-cong measure-pmf.measure-pmf-eq
      set-pair-pmf)
  moreover have ... =
    prob-space.prob (pair-pmf (pmf-of-set (Pow (V - {x}))) (pmf-of-set
      (Pow {x})))
      (Sigma E (λx. {y. even (card ((x ∪ y) ∩ ω)) = b}) ∩ (Pow (V -
        {x}) × Pow{x}))
    unfolding ** by auto
  moreover have ... =
    prob-space.prob (pair-pmf (pmf-of-set (Pow (V - {x}))) (pmf-of-set
      (Pow {x})))
      (Sigma E (λx. {y. even (card ((x ∪ y) ∩ ω)) = b}))
    by (smt (verit, best) 1 2 Int-iff Sigma-cong measure-pmf.measure-pmf-eq
      set-pair-pmf)
  moreover have ... =
    measure-pmf.prob (pmf-of-set (Pow (V - {x}))) E *
      (1 / 2)
    apply (subst measure-pmf-prob-dependent-product-bound-eq'[OF -
      - eR])
    by auto
  ultimately show ?thesis by auto
qed

```

```

lemma prob-random-xor-split:
  assumes V: finite V

```

**shows**  $\text{prob-space.prob } (\text{random-xor } V) E =$   
 $1 / 2 * \text{prob-space.prob } (\text{pmf-of-set } (\text{Pow } V)) \{e. (e, \text{True}) \in E\} +$   
 $1 / 2 * \text{prob-space.prob } (\text{pmf-of-set } (\text{Pow } V)) \{e. (e, \text{False}) \in E\}$   
**proof** –  
**have**  $\text{fin}: \text{finite } (\text{set-pmf } (\text{random-xor } V))$   
**by** ( $\text{simp add: } V \text{ set-pmf-random-xor}$ )  
  
**have**  $\text{fin2}: \text{finite } ((\lambda(x, y). (y, x)) - ' \text{set-pmf } (\text{random-xor } V))$   
**by**( $\text{auto intro!: finite-vimageI[OF fin] simp add: inj-def}$ )  
  
**have**  $\text{rw}: \{x. (x, b) \in E \wedge (x, b) \in \text{set-pmf } (\text{random-xor } V)\} =$   
 $\{x. (x, b) \in E\} \cap \text{set-pmf } (\text{pmf-of-set } (\text{Pow } V))$  **for**  $b$   
**by** ( $\text{auto simp add: } V \text{ set-pmf-random-xor Pow-not-empty}$ )  
  
**have**  $\text{prob-space.prob } (\text{random-xor } V) E =$   
 $\text{prob-space.prob } (\text{random-xor } V) (E \cap \text{set-pmf } (\text{random-xor } V))$   
**by** ( $\text{simp add: measure-Int-set-pmf}$ )  
  
**moreover have**  $\dots =$   
 $\text{measure-pmf.prob}$   
 $(\text{pair-pmf } (\text{bernoulli-pmf } (1 / 2)))$   
 $(\text{pmf-of-set } (\text{Pow } V))$   
 $((\lambda(x, y). (y, x)) - ' (E \cap \text{set-pmf } (\text{random-xor } V)))$   
**unfolding**  $\text{random-xor-def}$   
**apply** ( $\text{subst pair-commute-pmf}$ )  
**by**  $\text{simp}$   
**moreover have**  $\dots =$   
 $1 / 2 *$   
 $\text{measure-pmf.prob } (\text{pmf-of-set } (\text{Pow } V)) \{x. (x, \text{True}) \in E\} +$   
 $1 / 2 *$   
 $\text{measure-pmf.prob } (\text{pmf-of-set } (\text{Pow } V)) \{x. (x, \text{False}) \in E\}$   
**unfolding**  $\text{pair-pmf-def}$   
**apply** ( $\text{subst prob-bernoulli-bind-pmf}$ )  
**using**  $\text{fin2}$   
**unfolding**  $\text{map-pmf-def[symmetric] measure-map-pmf}$   
**by** ( $\text{auto simp add: vimage-def rw simp add: measure-Int-set-pmf}$ )  
**ultimately show**  $?thesis$  **by**  $\text{auto}$   
**qed**

**lemma**  $\text{prob-random-xor-nonempty-parity}$ :  
**assumes**  $V: \text{finite } V$   
**assumes**  $\omega: x \in \omega \ \omega \subseteq V$   
**assumes**  $E: \bigwedge c. c \in E \longleftrightarrow (\text{fst } c = \{x\}, \text{snd } c) \in E$   
**shows**  $\text{prob-space.prob } (\text{random-xor } V)$   
 $(E \cap \{c. \text{satisfies-xor } c \ \omega = b\}) =$   
 $1 / 2 * \text{prob-space.prob } (\text{random-xor } (V - \{x\})) E$   
**proof** –  
**have**  $*$ :  $\{e. (e, b') \in E \cap \{c. \text{satisfies-xor } c \ \omega = b\}\} =$   
 $\{e. (e, b') \in E\} \cap \{c. \text{even } (\text{card } (c \cap \omega)) = (b \neq b')\}$  **for**  $b'$

```

by (auto simp add: Int-commute)

have prob-space.prob (random-xor V)
  (E ∩ {c. satisfies-xor c ω = b}) =
  1 / 2 *
  measure-pmf.prob (pmf-of-set (Pow V))
  {e. (e, True) ∈ E ∩ {c. satisfies-xor c ω = b}} +
  1 / 2 *
  measure-pmf.prob (pmf-of-set (Pow V))
  {e. (e, False) ∈ E ∩ {c. satisfies-xor c ω = b}}
unfolding prob-random-xor-split[OF V] by auto
also have ... =
  1 / 2 *
  measure-pmf.prob (pmf-of-set (Pow V))
  ({e. (e, True) ∈ E} ∩ {c. even (card (c ∩ ω)) = (b ≠ True)}) +
  1 / 2 *
  measure-pmf.prob (pmf-of-set (Pow V))
  ({e. (e, False) ∈ E} ∩ {c. even (card (c ∩ ω)) = (b ≠ False)})
unfolding * by auto
also have ... =
  1 / 2 *
  (1 / 2 *
  measure-pmf.prob (pmf-of-set (Pow (V - {x})))
  {e. (e, True) ∈ E} +
  1 / 2 *
  measure-pmf.prob (pmf-of-set (Pow (V - {x})))
  {e. (e, False) ∈ E})
apply (subst prob-pmf-of-set-nonempty-parity[OF V ω])
subgoal using E by clarsimp
apply (subst prob-pmf-of-set-nonempty-parity[OF V ω])
using E by auto
also have ... =
  1 / 2 * measure-pmf.prob (random-xor (V - {x})) E
apply (subst prob-random-xor-split[symmetric])
using V by auto
finally show ?thesis by auto
qed

```

```

lemma pair-satisfies-random-xor-parity-1:
assumes V:finite V
assumes x: x ∉ ω x ∈ ω'
assumes ω: ω ⊆ V ω' ⊆ V
shows prob-space.prob (random-xor V)
  {c. satisfies-xor c ω = b ∧ satisfies-xor c ω' = b'} = 1 / 4
proof -
have wa: ω ∩ (a - {x}) = ω ∩ a for a
using x
by blast
have prob-space.prob (random-xor V)

```

```

    {c. satisfies-xor c ω = b ∧ satisfies-xor c ω' = b'} =
    prob-space.prob (random-xor V)
    ({c. satisfies-xor c ω = b} ∩ {c. satisfies-xor c ω' = b'})
    by (simp add: Collect-conj-eq)
  also have ... =
    1 / 2 *
    measure-pmf.prob (random-xor (V - {x})) {c. satisfies-xor c ω =
  b}
    apply (subst prob-random-xor-nonempty-parity[OF V x(2) ω(2)])
    by (auto simp add: wa)
  also have ... = 1/4
    apply (subst satisfies-random-xor-parity)
    using V by auto
  finally show ?thesis by auto
qed

```

```

lemma pair-satisfies-random-xor-parity:
  assumes V:finite V
  assumes ω: ω ≠ ω' ω ⊆ V ω' ⊆ V
  shows prob-space.prob (random-xor V)
    {c. satisfies-xor c ω = b ∧ satisfies-xor c ω' = b'} = 1 / 4
proof -
  obtain x where x ∉ ω ∧ x ∈ ω' ∨ x ∉ ω' ∧ x ∈ ω
    using ω
    by blast
  moreover {
    assume x: x ∉ ω x ∈ ω'
    have ?thesis using pair-satisfies-random-xor-parity-1[OF V x ω(2-3)]
      by blast
  }
  moreover {
    assume x: x ∉ ω' x ∈ ω
    then have ?thesis using pair-satisfies-random-xor-parity-1[OF V
  x ω(3) ω(2)]
      by (simp add: Collect-conj-eq Int-commute)
  }
  ultimately show ?thesis by auto
qed

```

```

lemma prob-pmf-of-set-nonempty-parity-UNIV:
  assumes finite V
  assumes x ∈ ω ω ⊆ V
  shows prob-space.prob (pmf-of-set (Pow V))
    {c. even (card (c ∩ ω)) = b} = 1 / 2
  using prob-pmf-of-set-nonempty-parity[OF assms, of UNIV]
  by auto

```

```

lemma prob-Pow-split:
  assumes ω ⊆ V finite V

```

**shows**  $\text{prob-space.prob (pmf-of-set (Pow V))}$   
 $\{x. P (\omega \cap x) \wedge Q ((V - \omega) \cap x)\} =$   
 $\text{prob-space.prob (pmf-of-set (Pow } \omega))$   
 $\{x. P x\} *$   
 $\text{prob-space.prob (pmf-of-set (Pow (V - } \omega))$   
 $\{x. Q x\}$

**proof** –  
**have** 1:  $\text{set-pmf (pmf-of-set (Pow } \omega)) = Pow \omega$   
**by** (*meson Pow-not-empty assms(1) assms(2) finite-Pow-iff finite-subset set-pmf-of-set*)  
**have** 2:  $\text{set-pmf (pmf-of-set (Pow (V - } \omega)) = Pow (V - \omega)$   
**by** (*simp add: Pow-not-empty assms(2)*)

**have** \*:  $(\text{pmf-of-set (Pow } \omega \times Pow (V - \omega))) =$   
 $(\text{pair-pmf (pmf-of-set (Pow } \omega)) (\text{pmf-of-set (Pow (V - } \omega))))$   
**unfolding** 1 2  
**apply** (*subst pmf-of-set-prod*)  
**using** *assms rev-finite-subset by auto*  
**have** \*\*:  $((\lambda(x, y). x \cup y) - ' \{x. P (\omega \cap x) \wedge Q ((V - \omega) \cap x)\} \cap ((Pow \omega) \times (Pow (V - \omega)))) =$   
 $(\{x. P x\} \cap Pow \omega) \times (\{x. Q x\} \cap Pow (V - \omega))$   
**apply** (*rule antisym*)  
**subgoal**  
**apply** *clarsimp*  
**by** (*smt (verit) Diff-disjoint Int-Un-eq(4) inf.orderE inf-commute inf-sup-distrib1 sup-bot.right-neutral sup-commute*)  
**apply** (*intro subsetI*)  
**apply** *clarsimp*  
**by** (*smt (verit, ccfv-threshold) Diff-Int Diff-disjoint Diff-empty Diff-eq-empty-iff Un-Int-assoc-eq Un-commute sup-bot.left-neutral*)

**have**  $\text{prob-space.prob (pmf-of-set (Pow V))}$   
 $\{x. P (\omega \cap x) \wedge Q ((V - \omega) \cap x)\} =$   
 $\text{prob-space.prob (map-pmf } (\lambda(x, y). x \cup y)$   
 $(\text{pmf-of-set (Pow } \omega \times Pow (V - \omega))))$   
 $\{x. P (\omega \cap x) \wedge Q ((V - \omega) \cap x)\}$   
**apply** (*subst pmf-of-set-powerset-split[symmetric, OF assms(1-2)]*)  
**by** *auto*

**moreover have** ... =  
 $\text{measure-pmf.prob (pair-pmf (pmf-of-set (Pow } \omega)) (\text{pmf-of-set (Pow (V - } \omega))))$   
 $((\lambda(x, y). x \cup y) - ' \{x. P (\omega \cap x) \wedge Q ((V - \omega) \cap x)\})$   
**unfolding** *measure-map-pmf*  
**using** \*  
**by** *presburger*

**moreover have** ... =

```

    measure-pmf.prob
    (pair-pmf (pmf-of-set (Pow  $\omega$ )) (pmf-of-set (Pow (V -  $\omega$ ))))
    ((( $\lambda(x, y). x \cup y$ ) - '
      { $x. P (\omega \cap x) \wedge Q ((V - \omega) \cap x)$ }))  $\cap$  ((Pow  $\omega$ )  $\times$  (Pow (V -
 $\omega$ ))))
    using 1 2
    by (smt (verit) Int-Collect Int-def Sigma-cong inf-idem measure-pmf.measure-pmf-eq
    set-pair-pmf)
    moreover have ... =
      measure-pmf.prob
      (pair-pmf (pmf-of-set (Pow  $\omega$ )) (pmf-of-set (Pow (V -  $\omega$ ))))
      (({ $x. P x$ }  $\cap$  Pow  $\omega$ )  $\times$  ({ $x. Q x$ }  $\cap$  Pow (V -  $\omega$ )))
    unfolding **
    by auto
    moreover have ... =
      measure-pmf.prob
      (pmf-of-set (Pow  $\omega$ )) ({ $x. P x$ }  $\cap$  Pow  $\omega$ ) *
      measure-pmf.prob
      (pmf-of-set (Pow (V -  $\omega$ )) ({ $x. Q x$ }  $\cap$  Pow (V -  $\omega$ )))
    apply (intro measure-pmf-product)
    subgoal by (meson assms(1) assms(2) countable-finite finite-Int
    finite-Pow-iff rev-finite-subset)
    by (simp add: assms(2) countable-finite)
    moreover have ... =
      measure-pmf.prob
      (pmf-of-set (Pow  $\omega$ )) { $x. P x$ } *
      measure-pmf.prob
      (pmf-of-set (Pow (V -  $\omega$ )) { $x. Q x$ })
    by (metis 1 2 measure-Int-set-pmf)
    ultimately show ?thesis by auto
qed

```

**lemma** *disjoint-prob-pmf-of-set-nonempty*:

```

    assumes  $\omega: x \in \omega \ \omega \subseteq V$ 
    assumes  $\omega': x' \in \omega' \ \omega' \subseteq V$ 
    assumes  $\omega \cap \omega' = \{\}$ 
    assumes  $V: \text{finite } V$ 
    shows prob-space.prob (pmf-of-set (Pow V))
      { $c. \text{even } (\text{card } (\omega \cap c)) = b \wedge \text{even } (\text{card } (\omega' \cap c)) = b'$ } = 1 / 4
  proof -
    have prob-space.prob (pmf-of-set (Pow V))
      { $c. \text{even } (\text{card } (\omega \cap c)) = b \wedge \text{even } (\text{card } (\omega' \cap c)) = b'$ } =
      prob-space.prob (pmf-of-set (Pow V))
      { $c. \text{even } (\text{card } (\omega \cap c)) = b \wedge \text{even } (\text{card } (((V - \omega) \cap c) \cap \omega'))$ }
    = b'
    by (smt (verit) Collect-cong Diff-Diff-Int Diff-eq-empty-iff Int-Diff
    Int-commute  $\omega'(2)$  assms(5) inf.orderE)

```

**moreover have** ... =  
*measure-pmf.prob (pmf-of-set (Pow  $\omega$ ))*  
*{x. even (card x) = b} \**  
*measure-pmf.prob (pmf-of-set (Pow (V -  $\omega$ )))*  
*{x. even (card (x  $\cap$   $\omega'$ ) = b'}*  
**apply** (*subst prob-Pow-split*)  
**using** *assms* **by** *auto*  
**moreover have** ... =  
*measure-pmf.prob (pmf-of-set (Pow  $\omega$ ))*  
*{x. even (card (x  $\cap$   $\omega$ ) = b} \**  
*measure-pmf.prob (pmf-of-set (Pow (V -  $\omega$ )))*  
*{x. even (card (x  $\cap$   $\omega'$ ) = b'}*  
**by** (*smt (verit, best) PowD Pow-not-empty  $\omega(2)$  assms(6) fi-*  
*nite-Pow-iff inf.orderE measure-pmf.measure-pmf-eq mem-Collect-eq*  
*rev-finite-subset set-pmf-of-set)*)

**moreover have** ... =  $1/4$   
**apply** (*subst prob-pmf-of-set-nonempty-parity-UNIV[OF -  $\omega(1)$ ]*)  
**subgoal using** *assms rev-finite-subset* **by** *blast*  
**subgoal by** *simp*  
**apply** (*subst prob-pmf-of-set-nonempty-parity-UNIV*)  
**using** *assms* **by** *auto*  
**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** *measure-pmf-prob-product-finite-set-pmf:*  
**assumes** *finite (set-pmf M) finite (set-pmf N)*  
**shows** *measure-pmf.prob (pair-pmf M N) (A  $\times$  B) =*  
*measure-pmf.prob M A \* measure-pmf.prob N B*  
**proof** –  
**have** *A: measure-pmf.prob M A = measure-pmf.prob M (A  $\cap$  set-pmf*  
*M)*  
**by** (*simp add: measure-Int-set-pmf*)  
**have** *B: measure-pmf.prob N B = measure-pmf.prob N (B  $\cap$  set-pmf*  
*N)*  
**by** (*simp add: measure-Int-set-pmf*)  
**have** *measure-pmf.prob M A \* measure-pmf.prob N B =*  
*measure-pmf.prob M (A  $\cap$  set-pmf M) \* measure-pmf.prob N (B*  
 *$\cap$  set-pmf N)*  
**using** *A B* **by** *auto*  
**moreover have** ... = *measure-pmf.prob (pair-pmf M N)*  
*((A  $\cap$  set-pmf M)  $\times$  (B  $\cap$  set-pmf N))*  
**apply** (*subst measure-pmf-prob-product[symmetric]*)  
**by** *auto*  
**moreover have** ... = *measure-pmf.prob (pair-pmf M N)*  
*((A  $\times$  B)  $\cap$  set-pmf (pair-pmf M N))*  
**by** (*simp add: Times-Int-Times*)  
**moreover have** ... = *measure-pmf.prob (pair-pmf M N)*  
*((A  $\times$  B) )*



**using** *measure-Int-set-pmf* **by** *blast*  
**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** *prob-random-xor-split-space*:

**assumes**  $\omega \subseteq V$  *finite*  $V$   
**shows** *prob-space.prob* (*random-xor*  $V$ )  
 $\{(x,b). P (\omega \cap x) b \wedge Q ((V - \omega) \cap x)\} =$   
*prob-space.prob* (*random-xor*  $\omega$ )  
 $\{(x,b). P x b\} *$   
*prob-space.prob* (*pmf-of-set* (*Pow* ( $V - \omega$ )))  
 $\{x. Q x\}$

**proof** –

**have** *d1*: *set-pmf* (*random-xor*  $\omega$ ) = *Pow*  $\omega \times UNIV$   
**by** (*metis* *assms*(1) *assms*(2) *infinite-super set-pmf-random-xor*)  
**have** *d2*: *set-pmf* (*pmf-of-set* (*Pow* ( $V - \omega$ ))) = *Pow* ( $V - \omega$ )  
**by** (*simp* *add*: *Pow-not-empty* *assms*(2))  
**have** *rhs*: *prob-space.prob* (*random-xor*  $\omega$ )  
 $\{(x,b). P x b\} *$   
*prob-space.prob* (*pmf-of-set* (*Pow* ( $V - \omega$ )))  
 $\{x. Q x\} =$   
*prob-space.prob* (*pair-pmf* (*random-xor*  $\omega$ ) (*pmf-of-set* (*Pow* ( $V -$   
 $\omega$ ))))  
 $(\{(x,b). P x b\} \times \{x. Q x\})$   
**apply** (*subst* *measure-pmf-prob-product-finite-set-pmf*)  
**subgoal** **by** (*metis* *Pow-def* *assms*(1) *assms*(2) *finite-Collect-subsets*  
*finite-SigmaI* *finite-code* *rev-finite-subset* *set-pmf-random-xor*)  
**using** *assms* **by** (*auto* *simp* *add*: *Pow-not-empty*)

**from** *pmf-of-set-powerset-split*[*OF* *assms*]

**have** *\**: *pmf-of-set* (*Pow*  $V$ ) =  
*map-pmf* ( $\lambda(x, y). x \cup y$ )  
(*pair-pmf* (*pmf-of-set* (*Pow*  $\omega$ )) (*pmf-of-set* (*Pow* ( $V - \omega$ ))))  
**by** (*metis* *Pow-not-empty* *assms*(1) *assms*(2) *finite-Diff* *finite-Pow-iff*  
*pmf-of-set-prod* *rev-finite-subset*)

**have** *\*\**: *random-xor*  $V =$   
*map-pmf* ( $\lambda((x,b),y). (x \cup y, b)$ ) (*pair-pmf* (*random-xor*  $\omega$ ) (*pmf-of-set*  
(*Pow* ( $V - \omega$ ))))  
**unfolding** *random-xor-def* *\**  
**including** *monad-normalisation*  
**by** (*auto* *simp* *add*: *pair-pmf-def* *map-pmf-def* *case-prod-unfold*)

**have** *prob-space.prob* (*random-xor*  $V$ )  $\{(x,b). P (\omega \cap x) b \wedge Q ((V$   
 $- \omega) \cap x)\} =$   
*measure-pmf.prob*  
(*pair-pmf* (*random-xor*  $\omega$ ) (*pmf-of-set* (*Pow* ( $V - \omega$ ))))  
 $\{y. P (\omega \cap (fst (fst y) \cup snd y)) (snd (fst y)) \wedge$   
 $Q ((V - \omega) \cap (fst (fst y) \cup snd y))\}$

```

unfolding **
by (auto simp add:case-prod-unfold)

moreover have ... =
  prob-space.prob (pair-pmf (random-xor  $\omega$ ) (pmf-of-set (Pow (V -
 $\omega$ ))))
    ({(x,b). P x b}  $\times$  {x. Q x})
    apply (intro measure-pmf.measure-pmf-eq[where p =pair-pmf
(random-xor  $\omega$ )
  (pmf-of-set (Pow (V -  $\omega$ )))]])
    subgoal by simp
    apply (clarsimp simp add: d1 d2)
    by (smt (verit, del-insts) Diff-disjoint Int-Diff boolean-algebra-cancel.sup0
inf.orderE inf-commute inf-sup-distrib1 sup-bot.left-neutral)
    ultimately show ?thesis using rhs
    by simp
qed

```

**lemma** three-disjoint-prob-random-xor-nonempty:

```

assumes  $\omega$ :  $\omega \neq \{\}$   $\omega \subseteq V$ 
assumes  $\omega'$ :  $\omega' \neq \{\}$   $\omega' \subseteq V$ 
assumes I:  $I \subseteq V$ 
assumes int:  $I \cap \omega = \{\}$   $I \cap \omega' = \{\}$   $\omega \cap \omega' = \{\}$ 
assumes V: finite V
shows prob-space.prob (random-xor V)
  {c. satisfies-xor c I = b  $\wedge$ 
  even (card ( $\omega \cap$  fst c)) = b'  $\wedge$ 
  even (card ( $\omega' \cap$  fst c)) = b''} = 1 / 8

```

**proof** –

```

have finI: finite I
  using V I
  using rev-finite-subset by blast
have finVI: finite (V - I)
  using V I
  using rev-finite-subset by blast
obtain x x' where x:  $x \in \omega$  x':  $x' \in \omega'$  using  $\omega$   $\omega'$ 
  by blast

```

```

have rw1: $\omega \cap ((V - I) \cap xx) = \omega \cap xx$  for xx
  by (metis Diff-Int-distrib2 Diff-empty  $\omega(2)$  assms(6) inf.absorb-iff2
inf-assoc inf-left-commute)

```

```

have rw2: $\omega' \cap ((V - I) \cap xx) = \omega' \cap xx$  for xx
  by (metis Diff-Int-distrib2 Diff-empty  $\omega'(2)$  assms(7) inf.absorb-iff2
inf-assoc inf-left-commute)

```

```

have prob-space.prob (random-xor V)
  {c. satisfies-xor c I = b  $\wedge$ 
  even (card ( $\omega \cap$  fst c)) = b'  $\wedge$ 
  even (card ( $\omega' \cap$  fst c)) = b''} = 1 / 8

```

```

    even (card ( $\omega' \cap \text{fst } c$ )) =  $b'$ } =
  prob-space.prob (random-xor V)
  {(x,bb). satisfies-xor (I  $\cap$  x, bb) I = b  $\wedge$ 
    even (card ( $\omega \cap ((V - I) \cap x)$ )) =  $b' \wedge$ 
    even (card ( $\omega' \cap ((V - I) \cap x)$ )) =  $b''$ }
apply (intro arg-cong[where f = prob-space.prob (random-xor V)])
unfolding rw1 rw2 satisfies-xor-inter
by (smt (verit) Collect-cong prod.collapse split-conv)

```

```

moreover have ... =
  measure-pmf.prob (random-xor I)
  {c. satisfies-xor c I = b} *
  measure-pmf.prob (pmf-of-set (Pow (V - I)))
  {x. even (card ( $\omega \cap x$ )) =  $b' \wedge$ 
    even (card ( $\omega' \cap x$ )) =  $b''$ }
apply (subst prob-random-xor-split-space[OF I V])
by (metis (no-types, lifting) Collect-cong case-prodE case-prodI2)
moreover have ... = 1 / 8
apply (subst satisfies-random-xor-parity[OF finI])
apply (subst disjoint-prob-pmf-of-set-nonempty)
using x  $\omega$   $\omega'$  int finVI by auto
ultimately show ?thesis by auto
qed

```

**lemma** three-disjoint-prob-pmf-of-set-nonempty:

```

assumes  $\omega$ :  $x \in \omega$   $\omega \subseteq V$ 
assumes  $\omega'$ :  $x' \in \omega'$   $\omega' \subseteq V$ 
assumes  $\omega''$ :  $x'' \in \omega''$   $\omega'' \subseteq V$ 
assumes int:  $\omega \cap \omega' = \{\}$   $\omega' \cap \omega'' = \{\}$   $\omega'' \cap \omega = \{\}$ 
assumes V: finite V
shows prob-space.prob (pmf-of-set (Pow V))
  {c. even (card ( $\omega \cap c$ )) = b  $\wedge$  even (card ( $\omega' \cap c$ )) =  $b' \wedge$  even
(card ( $\omega'' \cap c$ )) =  $b''$ } = 1 / 8
proof -
have prob-space.prob (pmf-of-set (Pow V))
  {c. even (card ( $\omega \cap c$ )) = b  $\wedge$  even (card ( $\omega' \cap c$ )) =  $b' \wedge$  even
(card ( $\omega'' \cap c$ )) =  $b''$ } =
  prob-space.prob (pmf-of-set (Pow V))
  {c. even (card ( $\omega \cap c$ )) = b  $\wedge$ 
    even (card (((V -  $\omega$ )  $\cap$  c)  $\cap$   $\omega'$ )) =  $b' \wedge$ 
    even (card (((V -  $\omega$ )  $\cap$  c)  $\cap$   $\omega''$ )) =  $b''$ }
by (smt (verit,best) Collect-cong Diff-Diff-Int Diff-eq-empty-iff
Int-Diff Int-commute  $\omega'$   $\omega''$  int inf.orderE)

```

```

moreover have ... =
  measure-pmf.prob (pmf-of-set (Pow  $\omega$ ))
  {x. even (card x) = b} *
  measure-pmf.prob (pmf-of-set (Pow (V -  $\omega$ )))

```

```

    {x. even (card ( $\omega' \cap x$ )) =  $b' \wedge$  even (card ( $\omega'' \cap x$ )) =  $b''$ }
  apply (subst prob-Pow-split)
  using assms by (auto simp add: inf commute)
  moreover have ... =
    measure-pmf.prob (pmf-of-set (Pow  $\omega$ ))
    {x. even (card ( $x \cap \omega$ )) =  $b$ } *
    measure-pmf.prob (pmf-of-set (Pow (V -  $\omega$ )))
    {x. even (card ( $\omega' \cap x$ )) =  $b' \wedge$  even (card ( $\omega'' \cap x$ )) =  $b''$ }
  by (smt (verit, best) PowD Pow-not-empty  $\omega$  V finite-Pow-iff
  inf.orderE measure-pmf.measure-pmf-eq mem-Collect-eq rev-finite-subset
  set-pmf-of-set)

```

```

  moreover have ... = 1/8
  apply (subst prob-pmf-of-set-nonempty-parity-UNIV[OF -  $\omega(1)$ ])
  subgoal using  $\omega(2)$  V rev-finite-subset by blast
  subgoal by simp
  apply (subst disjoint-prob-pmf-of-set-nonempty)
  using assms by auto
  ultimately show ?thesis by auto
qed

```

lemma four-disjoint-prob-random-xor-nonempty:

```

  assumes  $\omega$ :  $\omega \neq \{\}$   $\omega \subseteq V$ 
  assumes  $\omega'$ :  $\omega' \neq \{\}$   $\omega' \subseteq V$ 
  assumes  $\omega''$ :  $\omega'' \neq \{\}$   $\omega'' \subseteq V$ 
  assumes I:  $I \subseteq V$ 
  assumes int:  $I \cap \omega = \{\}$   $I \cap \omega' = \{\}$   $I \cap \omega'' = \{\}$ 
     $\omega \cap \omega' = \{\}$   $\omega' \cap \omega'' = \{\}$   $\omega'' \cap \omega = \{\}$ 
  assumes V: finite V
  shows prob-space.prob (random-xor V)
    {c. satisfies-xor c I =  $b0 \wedge$ 
      even (card ( $\omega \cap$  fst c)) =  $b \wedge$ 
      even (card ( $\omega' \cap$  fst c)) =  $b' \wedge$ 
      even (card ( $\omega'' \cap$  fst c)) =  $b''$ } = 1 / 16

```

proof –

```

  have finI: finite I
  using V I
  using rev-finite-subset by blast
  have finVI: finite (V - I)
  using V I
  using rev-finite-subset by blast
  obtain x x' x'' where x:  $x \in \omega$   $x' \in \omega'$   $x'' \in \omega''$ 
  using  $\omega$   $\omega'$   $\omega''$ 
  by blast

```

```

  have rw1:  $\omega \cap ((V - I) \cap xx) = \omega \cap xx$  for xx

```

```

  by (metis Diff-Int-distrib2 Diff-empty  $\omega(2)$  int(1) inf.absorb-iff2
  inf-assoc inf-left-commute)

```

**have**  $rw2:\omega' \cap ((V - I) \cap xx) = \omega' \cap xx$  **for**  $xx$   
**by** (*metis Diff-Int-distrib2 Diff-empty*  $\omega'(2)$  *int(2) inf. absorb-iff2*  
*inf-assoc inf-left-commute*)

**have**  $rw3:\omega'' \cap ((V - I) \cap xx) = \omega'' \cap xx$  **for**  $xx$   
**by** (*metis Diff-Int-distrib2 Diff-empty*  $\omega''(2)$  *int(3) inf. absorb-iff2*  
*inf-assoc inf-left-commute*)

**have** *prob-space.prob (random-xor V)*  
 $\{c. \text{satisfies-xor } c \ I = b0 \ \wedge$   
 $\text{even } (\text{card } (\omega \cap \text{fst } c)) = b \ \wedge$   
 $\text{even } (\text{card } (\omega' \cap \text{fst } c)) = b' \ \wedge$   
 $\text{even } (\text{card } (\omega'' \cap \text{fst } c)) = b''\} =$   
*prob-space.prob (random-xor V)*  
 $\{(x, bb). \text{satisfies-xor } (I \cap x, bb) \ I = b0 \ \wedge$   
 $\text{even } (\text{card } (\omega \cap ((V - I) \cap x))) = b \ \wedge$   
 $\text{even } (\text{card } (\omega' \cap ((V - I) \cap x))) = b' \ \wedge$   
 $\text{even } (\text{card } (\omega'' \cap ((V - I) \cap x))) = b''\}$   
**apply** (*intro arg-cong[where*  $f = \text{prob-space.prob (random-xor V)}$ ])  
**unfolding**  $rw1 \ rw2 \ rw3 \ \text{satisfies-xor-inter}$   
**by** (*smt (verit) Collect-cong prod.collapse split-conv*)

**moreover have**  $\dots =$   
*measure-pmf.prob (random-xor I)*  
 $\{c. \text{satisfies-xor } c \ I = b0\} *$   
*measure-pmf.prob (pmf-of-set (Pow (V - I)))*  
 $\{x. \text{even } (\text{card } (\omega \cap x)) = b \ \wedge$   
 $\text{even } (\text{card } (\omega' \cap x)) = b' \ \wedge$   
 $\text{even } (\text{card } (\omega'' \cap x)) = b''\}$   
**apply** (*subst prob-random-xor-split-space[OF I V]*)  
**by** (*metis (no-types, lifting) Collect-cong case-prodE case-prodI2*)  
**moreover have**  $\dots = 1 / 16$   
**apply** (*subst satisfies-random-xor-parity[OF finI]*)  
**apply** (*subst three-disjoint-prob-pmf-of-set-nonempty*)  
**using**  $x \ \omega \ \omega' \ \omega'' \ \text{int } \text{finVI}$  **by** *auto*  
**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** *three-satisfies-random-xor-parity-1:*

**assumes**  $V:\text{finite } V$   
**assumes**  $\omega: \omega \subseteq V \ \omega' \subseteq V \ \omega'' \subseteq V$   
**assumes**  $x: x \notin \omega \ x \notin \omega' \ x \in \omega''$   
**assumes**  $d: \omega \neq \omega'$   
**shows** *prob-space.prob (random-xor V)*  
 $\{c. \text{satisfies-xor } c \ \omega = b \ \wedge$   
 $\text{satisfies-xor } c \ \omega' = b' \ \wedge$   
 $\text{satisfies-xor } c \ \omega'' = b''\} = 1 / 8$

**proof** –

**have**  $wa: \omega \cap (a - \{x\}) = \omega \cap a$  **for**  $a$   
**using**  $x$   
**by** *blast*  
**have**  $wa': \omega' \cap (a - \{x\}) = \omega' \cap a$  **for**  $a$   
**using**  $x$   
**by** *blast*  
**have**  $prob\text{-}space.\text{prob} (random\text{-}xor V)$   
 $\{c.$   
 $\quad satisfies\text{-}xor c \omega = b \wedge satisfies\text{-}xor c \omega' = b' \wedge$   
 $\quad satisfies\text{-}xor c \omega'' = b''\}$  =  
 $prob\text{-}space.\text{prob} (random\text{-}xor V)$   
 $(\{c. satisfies\text{-}xor c \omega = b \wedge satisfies\text{-}xor c \omega' = b'\} \cap$   
 $\quad \{c. satisfies\text{-}xor c \omega'' = b''\})$   
**by** (*simp add: Collect-conj-eq inf-assoc*)  
**moreover have** ... =  
 $1 / 2 *$   
 $measure\text{-}pmf.\text{prob} (random\text{-}xor (V - \{x\}))$   
 $\{c. satisfies\text{-}xor c \omega = b \wedge satisfies\text{-}xor c \omega' = b'\}$   
**apply** (*subst prob-random-xor-nonempty-parity[OF V x(3)  $\omega(3)$ ]*)  
**by** (*auto simp add: wa wa'*)  
**moreover have** ... =  $1/8$   
**apply** (*subst pair-satisfies-random-xor-parity*)  
**using**  $V \omega x d$  **by** *auto*  
**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** *split-boolean-eq*:  
**shows**  $(A \longleftrightarrow B) = (b \longleftrightarrow I) \wedge$   
 $(B \longleftrightarrow C) = (b' \longleftrightarrow I) \wedge$   
 $(C \longleftrightarrow A) = (b'' \longleftrightarrow I)$   
 $\longleftrightarrow$   
 $I = odd(of\text{-}bool b + of\text{-}bool b' + of\text{-}bool b'') \wedge$   
 $(A = True \wedge$   
 $\quad B = (b' = b'') \wedge$   
 $\quad C = (b = b') \vee$   
 $\quad A = False \wedge$   
 $\quad B = (b' \neq b'') \wedge$   
 $\quad C = (b \neq b'))$   
**by** *auto*

**lemma** *three-satisfies-random-xor-parity*:  
**assumes**  $V:\text{finite } V$   
**assumes**  $\omega$ :  
 $\omega \neq \omega' \omega \neq \omega'' \omega' \neq \omega''$   
 $\omega \subseteq V \omega' \subseteq V \omega'' \subseteq V$   
**shows**  $prob\text{-}space.\text{prob} (random\text{-}xor V)$   
 $\{c. satisfies\text{-}xor c \omega = b \wedge$   
 $\quad satisfies\text{-}xor c \omega' = b' \wedge$   
 $\quad satisfies\text{-}xor c \omega'' = b''\} = 1 / 8$

```

proof –
  have ( $\exists x$ .
     $x \in \omega \wedge x \notin \omega' \wedge x \notin \omega'' \vee$ 
     $x \in \omega' \wedge x \notin \omega \wedge x \notin \omega'' \vee$ 
     $x \in \omega'' \wedge x \notin \omega \wedge x \notin \omega' \vee$ 
     $\omega - (\omega' \cup \omega'') = \{\} \wedge$ 
     $\omega' - (\omega \cup \omega'') = \{\} \wedge$ 
     $\omega'' - (\omega \cup \omega') = \{\}$ 
  ) by blast
  moreover {
    assume ( $\exists x$ .
       $x \in \omega \wedge x \notin \omega' \wedge x \notin \omega'' \vee$ 
       $x \in \omega' \wedge x \notin \omega \wedge x \notin \omega'' \vee$ 
       $x \in \omega'' \wedge x \notin \omega \wedge x \notin \omega'$ 
    )
    then obtain  $x$  where
       $x \in \omega \wedge x \notin \omega' \wedge x \notin \omega'' \vee$ 
       $x \in \omega' \wedge x \notin \omega \wedge x \notin \omega'' \vee$ 
       $x \in \omega'' \wedge x \notin \omega \wedge x \notin \omega'$  by auto
    moreover {
      assume  $x: x \notin \omega' \wedge x \notin \omega'' \wedge x \in \omega$ 
      have measure-pmf.prob (random-xor V)
        { $c$ . satisfies-xor c  $\omega' = b' \wedge$ 
          satisfies-xor c  $\omega'' = b'' \wedge$ 
          satisfies-xor c  $\omega = b$ } = 1 / 8
      apply (intro three-satisfies-random-xor-parity-1 [OF V - - - x])
      using  $\omega$  by auto
      then have ?thesis
      by (smt (verit, ccfv-SIG) Collect-cong)
    }
  }
  moreover {
    assume  $x: x \notin \omega \wedge x \notin \omega'' \wedge x \in \omega'$ 
    have measure-pmf.prob (random-xor V)
      { $c$ . satisfies-xor c  $\omega = b \wedge$ 
        satisfies-xor c  $\omega'' = b'' \wedge$ 
        satisfies-xor c  $\omega' = b'$ } = 1 / 8
    apply (intro three-satisfies-random-xor-parity-1 [OF V - - - x])
    using  $\omega$  by auto
    then have ?thesis
    by (smt (verit, ccfv-SIG) Collect-cong)
  }
  moreover {
    assume  $x: x \notin \omega \wedge x \notin \omega' \wedge x \in \omega''$ 
    have measure-pmf.prob (random-xor V)
      { $c$ . satisfies-xor c  $\omega = b \wedge$ 
        satisfies-xor c  $\omega' = b' \wedge$ 
        satisfies-xor c  $\omega'' = b''$ } = 1 / 8
    apply (intro three-satisfies-random-xor-parity-1 [OF V - - - x])
    using  $\omega$  by auto
    then have ?thesis
  }

```

```

    by (smt (verit, ccfv-SIG) Collect-cong)
  }
  ultimately have ?thesis by auto
}
moreover {
  assume dis:  $\omega - (\omega' \cup \omega'') = \{\}$   $\wedge$ 
     $\omega' - (\omega \cup \omega'') = \{\}$   $\wedge$ 
     $\omega'' - (\omega \cup \omega') = \{\}$ 

  define A where  $A = (\omega \cap \omega'') - \omega'$ 
  define B where  $B = (\omega \cap \omega') - \omega''$ 
  define C where  $C = (\omega' \cap \omega'') - \omega$ 
  define I where  $I = \omega \cap \omega' \cap \omega''$ 

  have f: finite A finite B finite C finite I
    unfolding A-def B-def C-def I-def
    by (meson V  $\omega$  finite-Diff finite-Int rev-finite-subset)+

  have s:  $A \subseteq V$   $B \subseteq V$   $C \subseteq V$   $I \subseteq V$ 
    unfolding A-def B-def C-def I-def
    using  $\omega$  by auto

  have i:  $I \cap A = \{\}$   $I \cap B = \{\}$   $I \cap C = \{\}$ 
     $B \cap C = \{\}$   $C \cap A = \{\}$   $A \cap B = \{\}$ 
    unfolding A-def B-def C-def I-def
    by blast +

  have s1:  $\omega = A \cup B \cup I$ 
    unfolding A-def B-def I-def
    using dis by auto
  have sx1: satisfies-xor (xx,bb)  $\omega = b \iff$ 
    even (card (A  $\cap$  xx)) = even (card (B  $\cap$  xx)) = (b  $\iff$  satis-
fies-xor(xx,bb) I) for xx bb
    unfolding s1 satisfies-xor.simps Int-Un-distrib2
    apply (subst card-Un-disjoint)
    subgoal using f by auto
    subgoal using f by auto
    subgoal using A-def B-def I-def by blast
    apply (subst card-Un-disjoint)
    subgoal using f by auto
    subgoal using f by auto
    subgoal using A-def B-def I-def by blast
    by auto

  have s2:  $\omega' = B \cup C \cup I$ 
    unfolding B-def C-def I-def
    using dis by auto
  have sx2: satisfies-xor (xx,bb)  $\omega' = b' \iff$ 
    even (card (B  $\cap$  xx)) = even (card (C  $\cap$  xx)) = (b'  $\iff$  satis-

```



```

fies-xor(xx,bb) I) for xx bb
  unfolding s2 satisfies-xor.simps Int-Un-distrib2
  apply (subst card-Un-disjoint)
  subgoal using f by auto
  subgoal using f by auto
  subgoal using B-def C-def I-def by blast
  apply (subst card-Un-disjoint)
  subgoal using f by auto
  subgoal using f by auto
  subgoal using B-def C-def I-def by blast
  by auto

have s3:  $\omega'' = A \cup C \cup I$ 
  unfolding A-def C-def I-def
  using dis by auto
have sx3: satisfies-xor (xx,bb)  $\omega'' = b'' \longleftrightarrow$ 
  even (card ( $C \cap xx$ )) = even (card ( $A \cap xx$ )) = ( $b'' \longleftrightarrow$  satis-
fies-xor(xx,bb) I) for xx bb
  unfolding s3 satisfies-xor.simps Int-Un-distrib2
  apply (subst card-Un-disjoint)
  subgoal using f by auto
  subgoal using f by auto
  subgoal using A-def B-def C-def I-def by blast
  apply (subst card-Un-disjoint)
  subgoal using f by auto
  subgoal using f by auto
  subgoal using A-def B-def C-def I-def by blast
  by auto

have  $A = \{\} \wedge B \neq \{\} \wedge C \neq \{\} \vee$ 
 $A \neq \{\} \wedge B = \{\} \wedge C \neq \{\} \vee$ 
 $A \neq \{\} \wedge B \neq \{\} \wedge C = \{\} \vee$ 
 $A \neq \{\} \wedge B \neq \{\} \wedge C \neq \{\}$ 
by (metis Un-commute  $\omega(1)$   $\omega(2)$   $\omega(3)$  s1 s2 s3)

moreover {
  assume as:  $A = \{\} B \neq \{\} C \neq \{\}$ 
  have satisfies-xor (xx,bb)  $\omega = b$ 
     $\wedge$  satisfies-xor (xx,bb)  $\omega' = b'$ 
     $\wedge$  satisfies-xor (xx,bb)  $\omega'' = b'' \longleftrightarrow$ 
satisfies-xor (xx, bb) I =
    odd (of-bool b + of-bool b' + of-bool b'')  $\wedge$ 
    (even (card ( $B \cap xx$ )) = ( $b' = b''$ )  $\wedge$ 
    even (card ( $C \cap xx$ )) = ( $b = b''$ )) for xx bb
  unfolding sx1 sx2 sx3
  apply(subst split-boolean-eq)
  using as by simp
  then have *: {c. satisfies-xor c  $\omega = b$ 
     $\wedge$  satisfies-xor c  $\omega' = b'$ 

```

```

     $\wedge \text{satisfies-xor } c \ \omega'' = b''\} =$ 
    {c. satisfies-xor c I =
      odd (of-bool b + of-bool b' + of-bool b'')  $\wedge$ 
      even (card (B  $\cap$  fst c)) = (b' = b'')  $\wedge$ 
      even (card (C  $\cap$  fst c)) = (b = b')\}
      by (smt (verit,best) Collect-cong prod.collapse)
have ?thesis
apply (subst *)
apply (intro three-disjoint-prob-random-xor-nonempty)
using as s i V by auto
}

```

```

moreover {
assume as: A  $\neq$  {} B  $\neq$  {} C = {}
have satisfies-xor (xx,bb)  $\omega'' = b''$ 
   $\wedge$  satisfies-xor (xx,bb)  $\omega = b$ 
   $\wedge$  satisfies-xor (xx,bb)  $\omega' = b' \longleftrightarrow$ 
  (satisfies-xor (xx, bb) I =
    odd (of-bool b'' + of-bool b + of-bool b')  $\wedge$ 
    (even (card (A  $\cap$  xx)) = (b = b')  $\wedge$ 
     even (card (B  $\cap$  xx)) = (b'' = b))) for xx bb
    unfolding sx1 sx2 sx3
    apply(subst split-boolean-eq)
    using as by simp
then have *: {c. satisfies-xor c  $\omega = b$ 
   $\wedge$  satisfies-xor c  $\omega' = b'$ 
   $\wedge$  satisfies-xor c  $\omega'' = b''\}$  =
  {c. satisfies-xor c I =
    odd (of-bool b'' + of-bool b + of-bool b')  $\wedge$ 
    (even (card (A  $\cap$  fst c)) = (b = b')  $\wedge$ 
     even (card (B  $\cap$  fst c)) = (b'' = b))\}
    by (smt (verit,best) Collect-cong prod.collapse)
have ?thesis
apply (subst *)
apply (intro three-disjoint-prob-random-xor-nonempty)
using as s i V by auto
}
}

```

```

moreover {
assume as: A  $\neq$  {} B = {} C  $\neq$  {}
have satisfies-xor (xx,bb)  $\omega' = b'$ 
   $\wedge$  satisfies-xor (xx,bb)  $\omega'' = b''$ 
   $\wedge$  satisfies-xor (xx,bb)  $\omega = b \longleftrightarrow$ 
  (satisfies-xor (xx, bb) I =
    odd (of-bool b' + of-bool b'' + of-bool b)  $\wedge$ 
    (even (card (C  $\cap$  xx)) = (b'' = b)  $\wedge$ 
     even (card (A  $\cap$  xx)) = (b' = b''))) for xx bb
    unfolding sx1 sx2 sx3
    apply(subst split-boolean-eq)

```

```

using as by simp
then have *: {c. satisfies-xor c ω = b
  ∧ satisfies-xor c ω' = b'
  ∧ satisfies-xor c ω'' = b''} =
{c. satisfies-xor c I =
  odd (of-bool b' + of-bool b'' + of-bool b) ∧
  (even (card (C ∩ fst c)) = (b'' = b) ∧
  even (card (A ∩ fst c)) = (b' = b''))}
by (smt (verit,best) Collect-cong prod.collapse)
have ?thesis
apply (subst *)
apply (intro three-disjoint-prob-random-xor-nonempty)
using as s i V by auto
}

```

```

moreover {
assume as: A ≠ {} B ≠ {} C ≠ {}
have 1: satisfies-xor (xx,bb) ω = b
  ∧ satisfies-xor (xx,bb) ω' = b'
  ∧ satisfies-xor (xx,bb) ω'' = b'' ↔
(satisfies-xor (xx, bb) I =
  odd (of-bool b + of-bool b' + of-bool b'') ∧
  even (card (A ∩ xx)) = True ∧
  even (card (B ∩ xx)) = (b' = b'') ∧
  even (card (C ∩ xx)) = (b = b') ∨
satisfies-xor (xx, bb) I =
  odd (of-bool b + of-bool b' + of-bool b'') ∧
  even (card (A ∩ xx)) = False ∧
  even (card (B ∩ xx)) = (b' ≠ b'') ∧
  even (card (C ∩ xx)) = (b ≠ b')) for xx bb
unfolding sx1 sx2 sx3
apply(subst split-boolean-eq)
by auto
have 2: satisfies-xor c ω = b
  ∧ satisfies-xor c ω' = b'
  ∧ satisfies-xor c ω'' = b'' ↔
(satisfies-xor c I =
  odd (of-bool b + of-bool b' + of-bool b'') ∧
  even (card (A ∩ fst c)) = True ∧
  even (card (B ∩ fst c)) = (b' = b'') ∧
  even (card (C ∩ fst c)) = (b = b') ∨
satisfies-xor c I =
  odd (of-bool b + of-bool b' + of-bool b'') ∧
  even (card (A ∩ fst c)) = False ∧
  even (card (B ∩ fst c)) = (b' ≠ b'') ∧
  even (card (C ∩ fst c)) = (b ≠ b')) for c
proof –
obtain xx bb where c:c = (xx,bb) by fastforce
show ?thesis unfolding c

```

```

    apply (subst 1)
    by auto
  qed
  have *: {c. satisfies-xor c ω = b
    ∧ satisfies-xor c ω' = b'
    ∧ satisfies-xor c ω'' = b''} =
  {c. satisfies-xor c I =
    odd (of-bool b + of-bool b' + of-bool b'') ∧
    even (card (A ∩ fst c)) = True ∧
    even (card (B ∩ fst c)) = (b' = b'') ∧
    even (card (C ∩ fst c)) = (b = b')} ∪
  {c. satisfies-xor c I =
    odd (of-bool b + of-bool b' + of-bool b'') ∧
    even (card (A ∩ fst c)) = False ∧
    even (card (B ∩ fst c)) = (b' ≠ b'') ∧
    even (card (C ∩ fst c)) = (b ≠ b')}
  apply (subst Un-def)
  apply (intro Collect-cong)
  apply (subst 2)
  by simp

  have **: 1 / 16 +
    measure-pmf.prob (random-xor V)
    {c. satisfies-xor c I =
      odd (of-bool b + of-bool b' +
        of-bool b'') ∧
      even (card (A ∩ fst c)) =
        False ∧
      even (card (B ∩ fst c)) =
        (b' ≠ b'') ∧
      even (card (C ∩ fst c)) =
        (b ≠ b')} =
    1 / 8
  apply (subst four-disjoint-prob-random-xor-nonempty)
  using as s i V by auto
  have ?thesis
  apply (subst *)
  apply (subst measure-pmf.finite-measure-Union)
  subgoal by simp
  subgoal by simp
  subgoal by auto
  apply (subst four-disjoint-prob-random-xor-nonempty)
  using as s i V ** by auto
}

ultimately have ?thesis by auto
}
ultimately show ?thesis by auto
qed

```

## 2.2 Independence for repeated XORs

We can lift the previous result to a list of independent sampled XORs.

**definition** *random-xors* :: 'a set  $\Rightarrow$  nat  $\Rightarrow$   
 (nat  $\rightarrow$  'a set  $\times$  bool) pmf  
**where** *random-xors* V n =  
 Pi-pmf {.. $n$ ::nat} None  
 ( $\lambda$ -. map-pmf Some (random-xor V))

**lemma** *random-xors-set*:

**assumes** V:finite V

**shows**

PiE-dflt {.. $n$ } None  
 (set-pmf  $\circ$  ( $\lambda$ -. map-pmf Some (random-xor V))) =  
 {xors. dom xors = {.. $n$ }  $\wedge$   
 ran xors  $\subseteq$  (Pow V)  $\times$  UNIV} (is ?lhs = ?rhs)

**proof** –

**have** ?lhs =

{f. dom f = {.. $n$ }  $\wedge$   
 ( $\forall x \in$  {.. $n$ }. f x  $\in$  Some ‘(Pow V  $\times$  UNIV))}

**unfolding** PiE-dflt-def o-def set-map-pmf set-pmf-random-xor[OF V]

**by** force

**also have** ... = ?rhs

**apply** (rule antisym)

**subgoal**

**apply** clarsimp

**by** (smt (z3) PowD domI image-iff mem-Collect-eq mem-Sigma-iff  
 option.simps(1) ran-def subsetD)

**apply** clarsimp

**by** (smt (verit, ccfv-threshold) domD image-iff lessThan-iff ranI  
 subsetD)

**finally show** ?thesis .

**qed**

**lemma** *random-xors-eq*:

**assumes** V:finite V

**shows** random-xors V n =

pmf-of-set

{xors. dom xors = {.. $n$ }  $\wedge$  ran xors  $\subseteq$  (Pow V)  $\times$  UNIV}

**proof** –

**have** pmf-of-set

{xors. dom xors = {.. $n$ }  $\wedge$   
 ran xors  $\subseteq$  (Pow V)  $\times$  UNIV} =

pmf-of-set

(PiE-dflt {.. $n$ } None)

```

      (set-pmf ∘ (λ-. map-pmf Some (random-xor V))))
unfolding random-xors-set[OF V] by auto
also have ... =
  Pi-pmf {.. $n$ } None
  (λx. pmf-of-set
    ((set-pmf ∘
      (λ-. map-pmf Some (random-xor V))) x))
apply (subst Pi-pmf-of-set[symmetric])
by (auto simp add:set-pmf-random-xor[OF V] V)
also have ... = random-xors V n
unfolding random-xors-def o-def set-map-pmf
apply (subst map-pmf-of-set-inj[symmetric])
subgoal by (auto simp add:set-pmf-random-xor[OF V] V)
subgoal by (auto simp add:set-pmf-random-xor[OF V] V)
subgoal by (auto simp add:set-pmf-random-xor[OF V] V)
by (metis V random-xor-pmf-of-set set-pmf-random-xor)
ultimately show ?thesis by auto
qed

```

```

definition xor-hash ::
  ('a → bool) ⇒
  (nat → ('a set × bool)) ⇒
  (nat → bool)
where xor-hash ω xors =
  (map-option
    (λxor. satisfies-xor xor {x. ω x = Some True}) ∘ xors)

```

```

lemma finite-map-set-nonempty:
assumes R ≠ {}
shows
  {xors.
    dom xors = D ∧ ran xors ⊆ R} ≠ {}
proof –
obtain r where r ∈ R
using assms by blast
then have (λx. if x ∈ D then Some r else None) ∈
  {xors. dom xors = D ∧ ran xors ⊆ R}
by (auto split:if-splits simp:ran-def)
thus ?thesis by auto
qed

```

```

lemma random-xors-set-pmf:
assumes V: finite V
shows
  set-pmf (random-xors V n) =
  {xors. dom xors = {.. $n$ } ∧
    ran xors ⊆ (Pow V) × UNIV}
unfolding random-xors-eq[OF V]

```

```

apply (intro set-pmf-of-set)
subgoal
  apply (intro finite-map-set-nonempty)
  by blast
apply (intro finite-set-of-finite-maps)
by (auto simp add: V)

```

```

lemma finite-random-xors-set-pmf:
  assumes V: finite V
  shows
    finite (set-pmf (random-xors V n))
  unfolding random-xors-set-pmf[OF V]
  by (auto intro!: finite-set-of-finite-maps simp add: V)

```

```

lemma map-eq-1:
  assumes dom f = dom g
  assumes  $\bigwedge x. x \in \text{dom } f \implies \text{the } (f x) = \text{the } (g x)$ 
  shows f = g
  by (metis assms(1) assms(2) domIff map-le-antisym map-le-def option.expand)

```

```

lemma xor-hash-eq-iff:
  assumes dom  $\alpha = \{..<n\}$ 
  shows xor-hash  $\omega x = \alpha \iff$ 
    (dom x =  $\{..<n\} \wedge$ 
      $(\forall i. i < n \implies$ 
       $(\exists \text{ xor}. x i = \text{Some } \text{xor} \wedge$ 
       satisfies-xor xor  $\{x. \omega x = \text{Some } \text{True}\} = \text{the } (\alpha i)$ 
      ))

```

```

proof -
  have 1: xor-hash  $\omega x = \alpha \iff$ 
    (dom (xor-hash  $\omega x$ ) = dom  $\alpha$ )  $\wedge$ 
     $(\forall i \in \text{dom } \alpha. \text{the } (\text{xor-hash } \omega x i) = \text{the } (\alpha i))$ 
  using map-eq-1 by fastforce
  have 2: dom (xor-hash  $\omega x$ ) = dom x
  unfolding xor-hash-def
  by auto
  have 3:  $\bigwedge i. i \in \text{dom } x \implies$ 
    xor-hash  $\omega x i = \text{Some } (\text{satisfies-xor } (\text{the } (x i)) \{x. \omega x = \text{Some } \text{True}\})$ 
  unfolding xor-hash-def
  by fastforce
  show ?thesis
  unfolding 1 assms 2
  using 3
  by (smt (verit, best) domD lessThan-iff option.sel)
qed

```

```

lemma xor-hash-eq-PiE-dfft:

```

```

assumes  $\text{dom } \alpha = \{..<n\}$ 
shows
   $\{xors. \text{xor-hash } \omega \text{ xors} = \alpha\} =$ 
   $\text{PiE-dflt } \{..<n\} \text{ None}$ 
   $(\lambda i. \text{Some } \langle$ 
     $\{xors. \text{satisfies-xor } xor \{x. \omega x = \text{Some True}\} = \text{the } (\alpha i)\rangle)$ 
proof –
  have *:  $\bigwedge x \text{ xa } a \text{ b.}$ 
     $(\neg \text{xa} < n \longrightarrow x \text{ xa} = \text{None}) \implies$ 
     $x \text{ xa} = \text{Some } (a, b) \implies \text{xa} < n$ 
  by (metis option.distinct(2))
  show ?thesis
  unfolding PiE-dflt-def
  unfolding xor-hash-eq-iff[OF assms]
  by (auto intro: * simp del: satisfies-xor.simps)
qed

```

**lemma** *prob-random-xors-xor-hash:*

```

assumes  $V: \text{finite } V$ 
assumes  $\alpha: \text{dom } \alpha = \{..<n\}$ 
shows
   $\text{measure-pmf.prob } (\text{random-xors } V \ n)$ 
   $\{xors. \text{xor-hash } \omega \text{ xors} = \alpha\} = 1 / 2 ^ n$ 
proof –
  have  $\text{measure-pmf.prob } (\text{random-xors } V \ n)$ 
     $\{xors. \text{xor-hash } \omega \text{ xors} = \alpha\} =$ 
     $\text{measure-pmf.prob}$ 
     $(\text{Pi-pmf } \{..<(n::\text{nat})\} \text{ None}$ 
       $(\lambda-. \text{map-pmf } \text{Some } (\text{random-xor } V)))$ 
     $(\text{PiE-dflt } \{..<n\} \text{ None}$ 
       $(\lambda i. \text{Some } \langle \{xors. \text{satisfies-xor } xor \{x. \omega x = \text{Some True}\} = \text{the}$ 
         $(\alpha i)\rangle\rangle))$ 
  unfolding random-xors-def xor-hash-eq-PiE-dflt[OF  $\alpha$ ]
  by auto
  also have ... =
     $(\prod x < n. \text{measure-pmf.prob } (\text{random-xor } V)$ 
       $(\{xors.$ 
         $\text{satisfies-xor } xor \{x. \omega x = \text{Some True}\} =$ 
         $\text{the } (\alpha x)\})$ 
    by (subst measure-Pi-pmf-PiE-dflt)
    (auto simp add: inj-vimage-image-eq)
  also have ... =  $(\prod x < n. 1/2)$ 
    by (simp add: assms(1) satisfies-random-xor-parity)
  also have ... =  $1 / 2 ^ n$ 
    by (simp add: power-one-over)
  finally show ?thesis by auto
qed

```

**lemma** *PiE-dflt-inter:*



**shows**  $PiE\text{-dflt } A \text{ dflt } B \cap PiE\text{-dflt } A \text{ dflt } B' =$   
 $PiE\text{-dflt } A \text{ dflt } (\lambda b. B \ b \cap B' \ b)$   
**unfolding**  $PiE\text{-dflt-def}$   
**by** *auto*

**lemma** *random-xors-xor-hash-pair*:

**assumes**  $V$ : *finite*  $V$

**assumes**  $\alpha$ :  $dom \ \alpha = \{..<n\}$

**assumes**  $\alpha'$ :  $dom \ \alpha' = \{..<n\}$

**assumes**  $\omega$ :  $dom \ \omega = V$

**assumes**  $\omega'$ :  $dom \ \omega' = V$

**assumes** *neq*:  $\omega \neq \omega'$

**shows**

$measure\text{-pmf}.\text{prob} \ (random\text{-xors} \ V \ n)$

$\{xors. \ xor\text{-hash} \ \omega \ xors = \alpha \wedge \ xor\text{-hash} \ \omega' \ xors = \alpha'\} =$   
 $1 / 4 \wedge n$

**proof** –

**obtain**  $y$  **where**  $\omega \ y \neq \omega' \ y$

**using** *neq*

**by** *blast*

**then have**  $neq:\{x. \ \omega \ x = \text{Some } True\} \neq \{x. \ \omega' \ x = \text{Some } True\}$

**by** (*smt* (*verit*, *ccfv-threshold*) *assms*(4) *assms*(5) *domD* *domIff*

*mem-Collect-eq*)

**have**  $measure\text{-pmf}.\text{prob} \ (random\text{-xors} \ V \ n)$

$\{xors.$

$\ xor\text{-hash} \ \omega \ xors = \alpha \wedge \ xor\text{-hash} \ \omega' \ xors = \alpha'\} =$

$measure\text{-pmf}.\text{prob} \ (random\text{-xors} \ V \ n)$

$(\{xors. \ xor\text{-hash} \ \omega \ xors = \alpha\} \cap$

$\{xors. \ xor\text{-hash} \ \omega' \ xors = \alpha'\})$

**by** (*simp* *add*: *Collect-conj-eq*)

**also have** ... =

$measure\text{-pmf}.\text{prob}$

$(Pi\text{-pmf} \ \{..<(n::nat)\} \ None$

$(\lambda-. \ map\text{-pmf} \ \text{Some} \ (random\text{-xor} \ V)))$

$(PiE\text{-dflt} \ \{..<n\} \ None$

$(\lambda i. \ \text{Some} \ ' \ {xor. \ satisfies\text{-xor} \ xor \ \{x. \ \omega \ x = \text{Some } True\} = the$

$(\alpha \ i)\})$

$\cap$

$PiE\text{-dflt} \ \{..<n\} \ None$

$(\lambda i. \ \text{Some} \ ' \ {xor. \ satisfies\text{-xor} \ xor \ \{x. \ \omega' \ x = \text{Some } True\} = the$

$(\alpha' \ i)\})$

**unfolding** *random-xors-def* *xor-hash-eq-PiE-dflt*[*OF*  $\alpha$ ] *xor-hash-eq-PiE-dflt*[*OF*

$\alpha'$ ]

**by** *auto*

**also have** ... =

$measure\text{-pmf}.\text{prob}$

$(Pi\text{-pmf} \ \{..<(n::nat)\} \ None$

$(\lambda-. \ map\text{-pmf} \ \text{Some} \ (random\text{-xor} \ V)))$

$(PiE\text{-dflt} \ \{..<n\} \ None$

```

( $\lambda i.$ 
  Some ' {xor.
    satisfies-xor xor {x.  $\omega$  x = Some True} = the ( $\alpha$  i)  $\wedge$ 
    satisfies-xor xor {x.  $\omega'$  x = Some True} = the ( $\alpha'$  i)}})
unfolding PiE-dflt-inter
apply (subst image-Int[symmetric])
by (auto simp add: Collect-conj-eq)

also have ... =
  ( $\prod x < n.$  measure-pmf.prob (random-xor V)
    {xor.
      satisfies-xor xor {x.  $\omega$  x = Some True} = the ( $\alpha$  x)  $\wedge$ 
      satisfies-xor xor {x.  $\omega'$  x = Some True} = the ( $\alpha'$  x)}})
by (subst measure-Pi-pmf-PiE-dflt)
  (auto simp add: inj-vimage-image-eq)
also have ... = ( $\prod x < n.$  1/4)
apply (subst pair-satisfies-random-xor-parity)
using assms neq by auto
also have ... = 1 / 4 ^ n
by (simp add: power-one-over)
finally show ?thesis by auto
qed

lemma random-xors-xor-hash-three:
assumes V: finite V
assumes  $\alpha$ : dom  $\alpha$  = {.. $n$ }
assumes  $\alpha'$ : dom  $\alpha'$  = {.. $n$ }
assumes  $\alpha''$ : dom  $\alpha''$  = {.. $n$ }
assumes  $\omega$ : dom  $\omega$  = V
assumes  $\omega'$ : dom  $\omega'$  = V
assumes  $\omega''$ : dom  $\omega''$  = V
assumes neq:  $\omega \neq \omega'$   $\omega' \neq \omega''$   $\omega'' \neq \omega$ 
shows
  measure-pmf.prob (random-xors V n)
    {xors.
      xor-hash  $\omega$  xors =  $\alpha$ 
       $\wedge$  xor-hash  $\omega'$  xors =  $\alpha'$ 
       $\wedge$  xor-hash  $\omega''$  xors =  $\alpha''$ } =
    1 / 8 ^ n
proof -
obtain x y z where  $\omega$  x  $\neq$   $\omega'$  x  $\omega'$  y  $\neq$   $\omega''$  y  $\omega''$  z  $\neq$   $\omega$  z
using neq
by blast
then have neq:
  {x.  $\omega$  x = Some True}  $\neq$  {x.  $\omega'$  x = Some True}
  {x.  $\omega'$  x = Some True}  $\neq$  {x.  $\omega''$  x = Some True}
  {x.  $\omega''$  x = Some True}  $\neq$  {x.  $\omega$  x = Some True}
by (smt (verit, ccfv-threshold) assms domD domIff mem-Collect-eq)+
have measure-pmf.prob (random-xors V n)

```

```

    {xor.
      xor-hash ω xors = α ∧ xor-hash ω' xors = α' ∧ xor-hash ω'' xors
= α''} =
  measure-pmf.prob (random-xors V n)
    ({xor. xor-hash ω xors = α} ∩
     {xor. xor-hash ω' xors = α'} ∩
     {xor. xor-hash ω'' xors = α''})
  by (simp add: measure-pmf.measure-pmf-eq)
moreover have ... =
  measure-pmf.prob
    (Pi-pmf {.. $n$ ::nat} None
     (λ-. map-pmf Some (random-xor V)))
    (PiE-dflt {.. $n$ } None
     (λi. Some ' {xor. satisfies-xor xor {x. ω x = Some True} = the
(α i)}))
    ∩
    (PiE-dflt {.. $n$ } None
     (λi. Some ' {xor. satisfies-xor xor {x. ω' x = Some True} = the
(α' i)}))
    ∩
    (PiE-dflt {.. $n$ } None
     (λi. Some ' {xor. satisfies-xor xor {x. ω'' x = Some True} = the
(α'' i)}))
  unfolding random-xors-def xor-hash-eq-PiE-dflt[OF α] xor-hash-eq-PiE-dflt[OF
α'] xor-hash-eq-PiE-dflt[OF α'']
  by auto
moreover have ... =
  measure-pmf.prob
    (Pi-pmf {.. $n$ ::nat} None
     (λ-. map-pmf Some (random-xor V)))
    (PiE-dflt {.. $n$ } None
     (λi.
       Some ' {xor.
         satisfies-xor xor {x. ω x = Some True} = the (α i) ∧
         satisfies-xor xor {x. ω' x = Some True} = the (α' i) ∧
         satisfies-xor xor {x. ω'' x = Some True} = the (α'' i)}))

unfolding PiE-dflt-inter
apply (subst image-Int[symmetric])
subgoal by simp
apply (intro arg-cong[where f=measure-pmf.prob
  (Pi-pmf {.. $n$ } None
   (λ-. map-pmf Some (random-xor V))))])
apply (intro arg-cong[where f=PiE-dflt {.. $n$ } None])
by auto

moreover have ... =
  (∏ x<n. measure-pmf.prob (random-xor V)
   {xor.

```

```

    satisfies-xor xor {x. ω x = Some True} = the (α x) ∧
    satisfies-xor xor {x. ω' x = Some True} = the (α' x) ∧
    satisfies-xor xor {x. ω'' x = Some True} = the (α'' x)))
  apply (subst measure-Pi-pmf-PiE-dflt)
  by (auto simp add: inj-vimage-image-eq)
moreover have ... = (∏ x<n. 1/8)
  apply (subst three-satisfies-random-xor-parity)
  subgoal using assms neq by clarsimp
  subgoal using assms neq by clarsimp
  subgoal using assms neq by clarsimp
  subgoal using assms neq by clarsimp
  subgoal using assms(5) by blast
  subgoal using assms(6) by blast
  subgoal using assms(7) by blast
  by auto
moreover have ... = 1 / 8 ^ n
  by (simp add: power-one-over)
ultimately show ?thesis by auto
qed

end

```

### 3 Random XOR hash family

This section defines a hash family based on random XORs and proves that this hash family is 3-universal.

```

theory RandomXORHashFamily imports
  RandomXOR
begin

```

```

lemma finite-dom:
  assumes finite V
  shows finite {w :: 'a → bool. dom w = V}
proof –
  have *: {w :: 'a → bool. dom w = V} =
    {w. dom w = V ∧ ran w ⊆ {True, False}}
  by auto
  show ?thesis unfolding *
  apply (intro finite-set-of-finite-maps)
  using assms by auto
qed

```

```

lemma xor-hash-eq-dom:
  assumes xor-hash ω xors = α
  shows dom xors = dom α
  using assms unfolding xor-hash-def
  by auto

```

**lemma** *prob-random-xors-xor-hash-indicat-real*:  
**assumes**  $V$ : *finite*  $V$   
**shows**  
 $\text{measure-pmf.prob } (\text{random-xors } V \ n)$   
 $\{xors. \text{xor-hash } \omega \ xors = \alpha\} =$   
 $\text{indicat-real } \{\alpha::\text{nat} \rightarrow \text{bool. dom } \alpha = \{0..<n\}\} \ \alpha /$   
 $\text{real } (\text{card } \{\alpha::\text{nat} \rightarrow \text{bool. dom } \alpha = \{0..<n\}\})$

**proof** –  
**have** \*:  $\{\alpha::\text{nat} \rightarrow \text{bool. dom } \alpha = \{0..<n\}\} =$   
 $\{\alpha. \text{dom } \alpha = \{0..<n\} \wedge \text{ran } \alpha \subseteq \{\text{True}, \text{False}\}\}$   
**by** *auto*  
**have** \*\*:  $\text{card } \{\alpha::\text{nat} \rightarrow \text{bool. dom } \alpha = \{0..<n\}\} = 2^n$   
**unfolding** \*  
**apply** (*subst card-dom-ran*)  
**by** (*auto simp add: numerals(2)*)  
**have**  $\text{dom } \alpha = \{..<n\} \vee \text{dom } \alpha \neq \{..<n\}$   
**by** *auto*  
**moreover** {  
**assume**  $\text{dom } \alpha = \{..<n\}$   
**from** *prob-random-xors-xor-hash[OF V this]*  
**have** *?thesis*  
**unfolding** \*\*  
**by** (*simp add: <dom alpha = {..<n}> atLeast0LessThan*)  
}

**moreover** {  
**assume** \*:  $\text{dom } \alpha \neq \{..<n\}$   
**then have**  $x \in \text{set-pmf } (\text{random-xors } V \ n) \implies$   
 $\alpha \neq \text{xor-hash } \omega \ x$  **for**  $x$   
**by** (*metis (mono-tags, lifting) V mem-Collect-eq random-xors-set-pmf xor-hash-eq-dom*)  
**then have**  $\text{measure-pmf.prob } (\text{random-xors } V \ n)$   
 $\{xors. \text{xor-hash } \omega \ xors = \alpha\} = 0$   
**apply** (*intro measure-pmf.measure-exclude[where A = set-pmf ((random-xors V n))]*)  
**by** (*auto simp add: Sigma-Algebra.measure-def emeasure-pmf xor-hash-eq-dom*)  
**then have** *?thesis*  
**by** (*simp add: \* atLeast0LessThan*)  
}

**ultimately show** *?thesis*  
**by** *auto*  
**qed**

**lemma** *xor-hash-family-uniform*:  
**assumes**  $V$ : *finite*  $V$   
**assumes**  $\text{dom } \omega = V$   
**shows** *prob-space.uniform-on*  
 $(\text{random-xors } V \ n)$   
 $(\text{xor-hash } i) \ \{\alpha. \text{dom } \alpha = \{0..<n\}\}$

```

apply (intro measure-pmf.uniform-onI[where  $p = \text{random-xors } V$ 
 $n$ ])
subgoal by clarsimp
subgoal using finite-dom by blast
using prob-random-xors-xor-hash-indicat-real[OF  $V$ ]
by (auto intro!: exI[where  $x = (\lambda i. \text{if } i < n \text{ then } \text{Some } \text{True} \text{ else } \text{None})$ ] split:if-splits)

```

**lemma** random-xors-xor-hash-pair-indicat:

```

assumes  $V$ : finite  $V$ 
assumes  $\omega$ : dom  $\omega = V$ 
assumes  $\omega'$ : dom  $\omega' = V$ 
assumes neq:  $\omega \neq \omega'$ 
shows
measure-pmf.prob (random-xors  $V$   $n$ )
  {xors.
    xor-hash  $\omega$  xors =  $\alpha \wedge \text{xor-hash } \omega' \text{ xors} = \alpha'$ } =
(measure-pmf.prob (random-xors  $V$   $n$ )
  {xors.
    xor-hash  $\omega$  xors =  $\alpha$ } *
measure-pmf.prob (random-xors  $V$   $n$ )
  {xors.
    xor-hash  $\omega' \text{ xors} = \alpha'$ })

```

**proof** –

```

have dom  $\alpha = \{..<n\} \wedge \text{dom } \alpha' = \{..<n\} \vee$ 
 $\neg(\text{dom } \alpha = \{..<n\} \wedge \text{dom } \alpha' = \{..<n\})$  by auto
moreover {
assume *: dom  $\alpha = \{..<n\} \wedge \text{dom } \alpha' = \{..<n\}$ 
have measure-pmf.prob (random-xors  $V$   $n$ )
  {xors.
    xor-hash  $\omega$  xors =  $\alpha \wedge \text{xor-hash } \omega' \text{ xors} = \alpha'$ } =  $1 / 4^{\wedge n}$ 
by (simp add: *(1) *(2) assms(1) assms(2) assms(3) assms(4))
random-xors-xor-hash-pair)

```

**moreover have**

```

measure-pmf.prob (random-xors  $V$   $n$ )
  {xors. xor-hash  $\omega$  xors =  $\alpha$ } =  $1/2^{\wedge n}$ 
by (simp add: *(1) assms(1) prob-random-xors-xor-hash)

```

**moreover have**

```

measure-pmf.prob (random-xors  $V$   $n$ )
  {xors. xor-hash  $\omega' \text{ xors} = \alpha'$ } =  $1/2^{\wedge n}$ 
by (simp add: *(2) assms(1) prob-random-xors-xor-hash)

```

**ultimately have** ?thesis

```

by (metis (full-types) Groups.mult-ac(2) four-x-squared power2-eq-square
power-mult power-one-over verit-prod-simplify(2))
}

```

**moreover** {

```

assume *: dom  $\alpha \neq \{..<n\} \vee \text{dom } \alpha' \neq \{..<n\}$ 
then have **:  $x \in \text{set-pmf } (\text{random-xors } V \text{ } n) \implies$ 
 $\alpha = \text{xor-hash } \omega \text{ } x \implies$ 

```

$\alpha' = \text{xor-hash } \omega' x \implies \text{False for } x$   
**by** (*metis* (*mono-tags*, *lifting*) *CollectD* *assms*(1) *random-xors-set-pmf*  
*xor-hash-eq-dom*)  
**have** *measure-pmf.prob* (*random-xors* *V n*)  
{*xors*.  
*xor-hash*  $\omega$  *xors* =  $\alpha$ } = 0  $\vee$   
*measure-pmf.prob* (*random-xors* *V n*)  
{*xors*.  
*xor-hash*  $\omega'$  *xors* =  $\alpha'$ } = 0  
**unfolding** *prob-random-xors-xor-hash-indicat-real*[*OF* *V*]  
**by** (*metis* (*full-types*) \* *atLeast0LessThan* *div-0* *indicator-simps*(2)  
*mem-Collect-eq*)  
**moreover** **have**  
*measure-pmf.prob* (*random-xors* *V n*)  
{*xors*.  
*xor-hash*  $\omega$  *xors* =  $\alpha$   $\wedge$  *xor-hash*  $\omega'$  *xors* =  $\alpha'$ } = 0  
**apply** (*intro* *measure-pmf.measure-exclude*[**where** *A* = *set-pmf*  
(*(random-xors* *V n*)]])  
**using** \*\* **by** (*auto simp add: Sigma-Algebra.measure-def* *emeasure-pmf*)  
**ultimately** **have** ?*thesis* **by** *auto*  
}  
**ultimately** **show** ?*thesis* **by** *auto*  
**qed**

**lemma** *prod-3-expand*:

**assumes**  $a \neq b$   $b \neq c$   $c \neq a$   
**shows** ( $\prod \omega \in \{a, b, c\}. f \omega$ ) =  $f a * (f b * f c)$   
**using** *assms* **by** *auto*

**lemma** *random-xors-xor-hash-three-indicat*:

**assumes** *V*: *finite* *V*  
**assumes**  $\omega$ : *dom*  $\omega$  = *V*  
**assumes**  $\omega'$ : *dom*  $\omega'$  = *V*  
**assumes**  $\omega''$ : *dom*  $\omega''$  = *V*  
**assumes** *neq*:  $\omega \neq \omega'$   $\omega' \neq \omega''$   $\omega'' \neq \omega$   
**shows**  
*measure-pmf.prob* (*random-xors* *V n*)  
{*xors*.  
*xor-hash*  $\omega$  *xors* =  $\alpha$   
 $\wedge$  *xor-hash*  $\omega'$  *xors* =  $\alpha'$   
 $\wedge$  *xor-hash*  $\omega''$  *xors* =  $\alpha''$ } =  
(*measure-pmf.prob* (*random-xors* *V n*)  
{*xors*.  
*xor-hash*  $\omega$  *xors* =  $\alpha$ } \*  
*measure-pmf.prob* (*random-xors* *V n*)  
{*xors*.  
*xor-hash*  $\omega'$  *xors* =  $\alpha'$ } \*  
*measure-pmf.prob* (*random-xors* *V n*)

```

    {xors.
      xor-hash ω'' xors = α''})
proof –
  have dom α = {.. $n$ } ∧ dom α' = {.. $n$ } ∧ dom α'' = {.. $n$ } ∨
    ¬(dom α = {.. $n$ } ∧ dom α' = {.. $n$ } ∧ dom α'' = {.. $n$ }) by
  auto
  moreover {
    assume *: dom α = {.. $n$ } dom α' = {.. $n$ } dom α'' = {.. $n$ }
    have 1:measure-pmf.prob (random-xors V n)
      {xors.
        xor-hash ω xors = α ∧
        xor-hash ω' xors = α' ∧
        xor-hash ω'' xors = α''} = 1 / 8 ^ n
      apply (intro random-xors-xor-hash-three)
      using V * ω ω' ω'' neq by auto
    have 2:
      measure-pmf.prob (random-xors V n)
        {xors. xor-hash ω xors = α} = 1/2 ^ n
      by (simp add: *(1) assms(1) prob-random-xors-xor-hash)
    have 3:
      measure-pmf.prob (random-xors V n)
        {xors. xor-hash ω' xors = α'} = 1/2 ^ n
      by (simp add: *(2) assms(1) prob-random-xors-xor-hash)
    have 4:
      measure-pmf.prob (random-xors V n)
        {xors. xor-hash ω'' xors = α''} = 1/2 ^ n
      by (simp add: *(3) assms(1) prob-random-xors-xor-hash)
    have ?thesis
      unfolding 1 2 3 4
      by (metis (mono-tags, opaque-lifting) arith-simps(11) arith-simps(12)
        arith-simps(58) divide-divide-eq-left mult.right-neutral power-mult-distrib
        times-divide-eq-right)
  }
  moreover {
    assume *: dom α ≠ {.. $n$ } ∨ dom α' ≠ {.. $n$ } ∨ dom α'' ≠
    {.. $n$ }
    then have **: x ∈ set-pmf (random-xors V n) ⇒
      α = xor-hash ω x ⇒
      α' = xor-hash ω' x ⇒
      α'' = xor-hash ω'' x ⇒ False for x
    by (metis (mono-tags, lifting) CollectD assms(1) random-xors-set-pmf
      xor-hash-eq-dom)
    have measure-pmf.prob (random-xors V n)
      {xors.
        xor-hash ω xors = α} = 0 ∨
      measure-pmf.prob (random-xors V n)
        {xors.
          xor-hash ω' xors = α'} = 0 ∨
      measure-pmf.prob (random-xors V n)

```



```

    {xors.
      xor-hash  $\omega''$  xors =  $\alpha''$ } = 0
    unfolding prob-random-xors-xor-hash-indicat-real[OF V]
    by (metis (full-types) * atLeast0LessThan div-0 indicator-simps(2)
mem-Collect-eq)
    moreover have
      measure-pmf.prob (random-xors V n)
      {xors.
        xor-hash  $\omega$  xors =  $\alpha$   $\wedge$  xor-hash  $\omega'$  xors =  $\alpha'$   $\wedge$  xor-hash  $\omega''$  xors
        =  $\alpha''$ } = 0
    apply (intro measure-pmf.measure-exclude[where A = set-pmf
((random-xors V n))])
    using ** by (auto simp add: Sigma-Algebra.measure-def emea-
sure-pmf)
    ultimately have ?thesis by auto
  }
  ultimately show ?thesis
  by fastforce
qed

```

**lemma** xor-hash-3-indep:

```

assumes V: finite V
assumes J: card J  $\leq$  3 J  $\subseteq$  { $\alpha$ . dom  $\alpha$  = V}
shows
  measure-pmf.prob (random-xors V n)
  {xors.  $\forall \omega \in J$ . xor-hash  $\omega$  xors = f  $\omega$ } =
  ( $\prod \omega \in J$ .
    measure-pmf.prob (random-xors V n)
    {xors. xor-hash  $\omega$  xors = f  $\omega$ })

```

**proof** –

```

have card J = 0  $\vee$  card J = 1  $\vee$  card J = 2  $\vee$  card J = 3
using assms by auto
moreover {
  assume card J = 0
  then have J = {}
  by (meson assms(1) assms(3) card-eq-0-iff finite-dom finite-subset)
  then have ?thesis
  by clarsimp
}
moreover {
  assume card J = 1
  then obtain x where J = {x}
  using card-1-singletonE by blast
  then have ?thesis
  by auto
}
moreover {
  assume card J = 2
  then obtain  $\omega$   $\omega'$  where J: J = { $\omega$ ,  $\omega'$ } and

```

```

     $\omega: \omega \neq \omega' \text{ dom } \omega = V \text{ dom } \omega' = V$ 
    unfolding card-2-iff
    using J
    by force
    have ?thesis unfolding J
      by (auto simp add: random-xors-xor-hash-pair-indicat V  $\omega$ )
  }
  moreover {
    assume card J = 3
    then obtain  $\omega \omega' \omega''$  where  $J:J = \{\omega, \omega', \omega''\}$  and
       $\omega: \omega \neq \omega' \omega' \neq \omega'' \omega'' \neq \omega$ 
       $\text{dom } \omega = V \text{ dom } \omega' = V \text{ dom } \omega'' = V$ 
    unfolding card-3-iff
    using J
    by force
    have ?thesis unfolding J
      by (auto simp add: random-xors-xor-hash-three-indicat V  $\omega$ 
        prod-3-expand[OF  $\omega(1-3)$ ])
  }
  ultimately show ?thesis by auto
qed

```

**lemma** *xor-hash-3-wise-indep*:

```

  assumes finite V
  shows prob-space.k-wise-indep-vars
    (random-xors V n) 3
    ( $\lambda$ -. Universal-Hash-Families-More-Independent-Families.discrete)
  xor-hash
    { $\alpha$ . dom  $\alpha = V$ }
  apply (subst prob-space.k-wise-indep-vars-def)
  by (auto intro!: measure-pmf.indep-vars-pmf xor-hash-3-indep simp
    add: measure-pmf.prob-space-axioms assms card-mono dual-order.trans)

```

**theorem** *xor-hash-family-3-universal*:

```

  assumes finite V
  shows prob-space.k-universal
    (random-xors V n) 3 xor-hash
    { $\alpha::a \rightarrow \text{bool}$ . dom  $\alpha = V$ }
    { $\alpha::\text{nat} \rightarrow \text{bool}$ . dom  $\alpha = \{0..<n\}$ }
  apply (subst prob-space.k-universal-def)
  subgoal by (clarsimp simp add: measure-pmf.prob-space-axioms )
  using xor-hash-3-wise-indep assms xor-hash-family-uniform assms
  by blast

```

**corollary** *xor-hash-family-2-universal*:

```

  assumes finite V
  shows prob-space.k-universal
    (random-xors V n) 2 xor-hash
    { $\alpha::a \rightarrow \text{bool}$ . dom  $\alpha = V$ }

```

```

    { $\alpha::nat \rightarrow bool. dom \alpha = \{0..<n\}$ }
  using assms
  by (auto intro!: prob-space.k-universal-mono[OF - - xor-hash-family-3-universal]
measure-pmf.prob-space-axioms)

```

end

## 4 ApproxMCCore definitions

This section defines the ApproxMCCore locale and various failure events to be used in its probabilistic analysis. The definitions closely follow Section 4.2 of Chakraborty et al. [1]. Some non-probabilistic properties of the events are proved, most notably, the event inclusions of Lemma 3 [1]. Note that “events” here refer to subsets of hash functions.

```

theory ApproxMCCore imports
  ApproxMCPreliminaries
begin

```

```

type-synonym 'a assg = 'a  $\rightarrow$  bool

```

```

definition restr :: 'a set  $\Rightarrow$  ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a assg
  where restr S w = ( $\lambda x. if\ x \in S\ then\ Some\ (w\ x)\ else\ None$ )

```

```

lemma restrict-eq-mono:

```

```

  assumes  $x \subseteq y$ 
  assumes  $f \mid' y = g \mid' y$ 
  shows  $f \mid' x = g \mid' x$ 
  using assms
  by (metis Map.restrict-restrict inf.absorb-iff2)

```

```

definition proj :: 'a set  $\Rightarrow$  ('a  $\Rightarrow$  bool) set  $\Rightarrow$  'a assg set
  where proj S W = restr S ' W

```

```

lemma card-proj:

```

```

  assumes finite S
  shows finite (proj S W)  $card\ (proj\ S\ W) \leq 2 \wedge card\ S$ 

```

```

proof -

```

```

  have *: proj S W  $\subseteq$  {w. dom w = S}

```

```

    unfolding proj-def restr-def

```

```

    by (auto split: if-splits)

```

```

  also have 1: {w. dom w = S} = {w. dom w = S  $\wedge$  ran w  $\subseteq$ 
    {True,False}}

```

```

    by auto

```

```

have f: finite {w. dom w = S ∧ ran w ⊆ {True,False}}
  by (simp add: assms finite-set-of-finite-maps)
thus finite (proj S W)
  using * 1
  by (metis finite-subset)

have 2:card {w. dom w = S ∧ ran w ⊆ {True,False}} = (card
{True,False}) ^ card S
  apply (intro card-dom-ran)
  using assms by auto

show card (proj S W) ≤ 2 ^ card S
  using * 1 2
  by (metis (no-types, lifting) f card.empty card-insert-disjoint card-mono
finite.emptyI finite.insertI insert-absorb insert-not-empty numeral-2-eq-2
singleton-insert-inj-eq)
qed

lemma proj-mono:
  assumes x ⊆ y
  shows proj w x ⊆ proj w y
  unfolding proj-def
  using assms by blast

definition aslice :: nat ⇒ nat assg ⇒ nat assg
  where aslice i a = a |' {..}

lemma aslice-eq:
  assumes i ≥ n
  assumes dom a = {..n}
  shows aslice i a = aslice n a
  using assms
  unfolding aslice-def restrict-map-def dom-def
  by fastforce

definition hslice :: nat ⇒
('a assg ⇒ nat assg) ⇒ ('a assg ⇒ nat assg)
  where hslice i h = aslice i ∘ h

locale ApproxMCCore =
  fixes W :: ('a ⇒ bool) set
  fixes S :: 'a set
  fixes ε :: real
  fixes α :: nat assg
  fixes thresh :: nat
  assumes α: dom α = {0..card S - 1}

```

```

assumes  $\varepsilon: \varepsilon > 0$ 
assumes thresh:
  thresh > 4
   $\text{card } (\text{proj } S \ W) \geq \text{thresh}$ 
assumes S: finite S
begin

lemma finite-proj-S:
  shows finite (proj S W)
  using S by (auto intro!: card-proj)

definition  $\mu :: \text{nat} \Rightarrow \text{real}$ 
  where  $\mu \ i = \text{card } (\text{proj } S \ W) / 2 \wedge i$ 

definition card-slice ::
  (a assg  $\Rightarrow$  nat assg)  $\Rightarrow$ 
  nat  $\Rightarrow$  nat
  where card-slice h i =
     $\text{card } (\text{proj } S \ W \cap \{w. \text{hslice } i \ h \ w = \text{aslice } i \ \alpha\})$ 

lemma card-slice-anti-mono:
  assumes  $i \leq j$ 
  shows card-slice h j  $\leq$  card-slice h i
proof –
  have  $*$ :  $\{..<i\} \subseteq \{..<j\}$  using assms by auto
  have  $\{w. \text{hslice } j \ h \ w = \text{aslice } j \ \alpha\}$ 
     $\subseteq \{w. \text{hslice } i \ h \ w = \text{aslice } i \ \alpha\}$ 
  by (auto intro: restrict-eq-mono[OF  $*$ ] simp add: hslice-def aslice-def)
  thus ?thesis
    unfolding card-slice-def
    apply (intro card-mono)
    subgoal using finite-proj-S by blast
    by (auto intro!: proj-mono)
qed

lemma hslice-eq:
  assumes  $n \leq i$ 
  assumes  $\bigwedge w. \text{dom } (h \ w) = \{..<n\}$ 
  shows hslice i h = hslice n h
  using assms aslice-eq
  unfolding hslice-def by auto

lemma card-slice-lim:
  assumes  $\text{card } S - 1 \leq i$ 
  assumes  $\bigwedge w. \text{dom } (h \ w) = \{..<(\text{card } S - 1)\}$ 
  shows card-slice h i = card-slice h (card S - 1)
  unfolding card-slice-def
  apply(subst aslice-eq[OF assms(1)])

```

**subgoal using  $\alpha$  by auto**  
**apply**(subst hslice-eq[OF assms(1)])  
**using assms by auto**

**definition**  $T :: \text{nat} \Rightarrow$   
 $(!a \text{ assg} \Rightarrow \text{nat assg}) \text{ set}$   
**where**  $T i = \{h. \text{card-slice } h \ i < \text{thresh}\}$

**lemma**  $T\text{-mono}$ :  
**assumes**  $i \leq j$   
**shows**  $T i \subseteq T j$   
**unfolding**  $T\text{-def}$   
**using**  $\text{card-slice-anti-mono}$ [OF assms]  
 $\text{dual-order.strict-trans2}$   
 $\text{of-nat-mono}$   
**by** blast

**lemma**  $\mu\text{-anti-mono}$ :  
**assumes**  $i \leq j$   
**shows**  $\mu i \geq \mu j$   
**proof** –  
**have**  $2^i \leq (2::\text{real})^j$   
**by** (simp add: assms)  
**then show** ?thesis **unfolding**  $\mu\text{-def}$   
**by** (simp add: frac-le)  
**qed**

**lemma**  $\text{card-proj-witnesses}$ :  
 $\text{card } (\text{proj } S \ W) > 0$   
**using** thresh **by** linarith

**lemma**  $\mu\text{-strict-anti-mono}$ :  
**assumes**  $i < j$   
**shows**  $\mu i > \mu j$   
**proof** –  
**have**  $2^i < (2::\text{real})^j$   
**by** (simp add: assms)  
**then show** ?thesis **unfolding**  $\mu\text{-def}$   
**using**  $\text{card-proj-witnesses}$   
**by** (simp add: frac-less2)  
**qed**

**lemma**  $\mu\text{-gt-zero}$ :  
**shows**  $\mu i > 0$   
**unfolding**  $\mu\text{-def}$   
**using**  $\text{card-proj-witnesses}$   
**by** auto

**definition**  $L :: nat \Rightarrow$   
 $(!a \text{ assg} \Rightarrow nat \text{ assg}) \text{ set}$   
**where**  
 $L \ i =$   
 $\{h. \text{real} (\text{card-slice } h \ i) < \mu \ i / (1 + \varepsilon)\}$

**definition**  $U :: nat \Rightarrow$   
 $(!a \text{ assg} \Rightarrow nat \text{ assg}) \text{ set}$   
**where**  
 $U \ i =$   
 $\{h. \text{real} (\text{card-slice } h \ i) \geq \mu \ i * (1 + \varepsilon / (1 + \varepsilon))\}$

**definition**  $\text{approxcore} ::$   
 $(!a \text{ assg} \Rightarrow nat \text{ assg}) \Rightarrow$   
 $nat \times nat$   
**where**  
 $\text{approxcore } h =$   
 $(\text{case } List.find$   
 $(\lambda i. h \in T \ i) [1..<card \ S] \text{ of}$   
 $None \Rightarrow (2 \wedge card \ S, 1)$   
 $| Some \ m \Rightarrow$   
 $(2 \wedge m, \text{card-slice } h \ m))$

**definition**  $\text{approxcore-fail} ::$   
 $(!a \text{ assg} \Rightarrow nat \text{ assg}) \text{ set}$   
**where**  $\text{approxcore-fail} =$   
 $\{h.$   
 $\text{let } (cells, sols) = \text{approxcore } h \text{ in}$   
 $cells * sols \notin$   
 $\{ \text{card} (\text{proj } S \ W) / (1 + \varepsilon) ..$   
 $(1 + \varepsilon::real) * \text{card} (\text{proj } S \ W)\}$   
 $\}$

**lemma**  $T0\text{-empty}$ :  
**shows**  $T \ 0 = \{\}$   
**unfolding**  $T\text{-def}$   $\text{card-slice-def}$   
 $hslice\text{-def}$   $aslice\text{-def}$   
**using**  $\text{thresh}(2)$  **by**  $\text{auto}$

**lemma**  $L0\text{-empty}$ :  
**shows**  $L \ 0 = \{\}$   
**proof** –  
**have**  $0 < \varepsilon \Longrightarrow$   
 $\text{real} (\text{card} (\text{proj } S \ W))$   
 $< \text{real} (\text{card} (\text{proj } S \ W)) / (1 + \varepsilon) \Longrightarrow$   
 $\text{False}$

**by** (*smt* (*z3*) *card-proj-witnesses divide-minus1 frac-less2 nonzero-minus-divide-right of-nat-0-less-iff*)  
**thus** *?thesis*  
**unfolding** *L-def card-slice-def hslice-def aslice-def μ-def*  
**using**  $\varepsilon$  **by** *clarsimp*  
**qed**

**lemma** *U0-empty:*  
**shows**  $U\ 0 = \{\}$   
**proof** –  
**have** \*:  $(1 + \varepsilon / (1 + \varepsilon)) > 1$   
**using**  $\varepsilon$  **by** *auto*  
**have** \*\*:  $U\ 0 = \{\}$   
**unfolding** *U-def card-slice-def hslice-def aslice-def μ-def*  
**using** \*  
**by** (*simp add: card-proj-witnesses*)  
**thus** *?thesis using \*\* by auto*  
**qed**

**lemma** *real-divide-pos-left:*  
**assumes**  $(0::real) < a$   
**assumes**  $a * b < c$   
**shows**  $b < c / a$   
**using** *assms*  
**by** (*simp add: mult.commute mult-imp-less-div-pos*)

**lemma** *real-divide-pos-right:*  
**assumes**  $a > (0::real)$   
**assumes**  $b < a * c$   
**shows**  $b / a < c$   
**using** *assms*  
**by** (*simp add: mult.commute mult-imp-div-pos-less*)

**lemma** *failure-imp:*  
**shows** *approxcore-fail*  $\subseteq$   
 $(\bigcup_{i \in \{1..<card\ S\}} (T\ i - T\ (i-1)) \cap (L\ i \cup U\ i)) \cup$   
 $-T\ (card\ S - 1)$   
**proof** *standard*  
**fix** *h*  
**assume**  $h \in \text{approxcore-fail}$   
**then obtain** *cells sols* **where**  
 $h: \text{approxcore } h = (\text{cells}, \text{sols})$   
 $\text{cells} * \text{sols} \notin$   
 $\{ \text{card } (\text{proj } S\ W) / (1 + \varepsilon) ..$   
 $(1 + \varepsilon::real) * \text{card } (\text{proj } S\ W) \}$  (**is** -  $\notin$   $\{?lower..?upper\}$ )



```

unfolding approxcore-fail-def
by auto

have List.find ( $\lambda i. h \in T i$ ) [ $1..<card S$ ] = None  $\vee$ 
  List.find ( $\lambda i. h \in T i$ ) [ $1..<card S$ ]  $\neq$  None by auto
moreover {
  assume List.find ( $\lambda i. h \in T i$ ) [ $1..<card S$ ] = None
  then have  $h \notin T (card S - 1)$ 
    unfolding find-None-iff
    by (metis T0-empty atLeastLessThan-iff diff-is-0-eq' diff-less
empty-iff gr-zeroI leI less-one not-less-zero set-upt)
  }
moreover {
  assume List.find ( $\lambda i. h \in T i$ ) [ $1..<card S$ ]  $\neq$  None
  then obtain m where
    findm: List.find ( $\lambda i. h \in T i$ ) [ $1..<card S$ ] = Some m by auto
  then have
     $m: 1 \leq m \wedge m < card S$ 
     $h \in T m$ 
     $\forall i. 1 \leq i \wedge i < m \longrightarrow h \notin T i$ 
    unfolding find-Some-iff
    using less-Suc-eq-0-disj by auto

then have  $1: h \notin T (m - 1)$ 
by (metis T0-empty bot-nat-0.not-eq-extremum diff-is-0-eq diff-less
empty-iff less-one)

have  $2^m * card\text{-}slice\ h\ m < ?lower \vee$ 
   $2^m * card\text{-}slice\ h\ m > ?upper$ 
using h unfolding approxcore-def findm by auto
moreover {
  assume  $2^m * card\text{-}slice\ h\ m < ?lower$ 
  then have  $card\text{-}slice\ h\ m < ?lower / 2^m$ 
    using real-divide-pos-left
  by (metis numeral-power-eq-of-nat-cancel-iff of-nat-mult zero-less-numeral
zero-less-power)

then have  $h \in L m$  unfolding L-def
  by (simp add:  $\mu$ -def mult.commute)
}
moreover {
assume  $?: ?upper < 2^m * card\text{-}slice\ h\ m$ 
have  $1 / (1 + \varepsilon) < 1$ 
using  $\varepsilon$  divide-less-eq-1 by fastforce
then have  $\varepsilon / (1 + \varepsilon) < \varepsilon$ 
using  $\varepsilon$ 
by (metis (no-types, opaque-lifting) add-cancel-left-right div-by-1
divide-divide-eq-left divide-less-eq-1-pos mult.commute nonzero-divide-mult-cancel-left)
then have  $card (proj\ S\ W) * (1 + \varepsilon / (1 + \varepsilon)) \leq ?upper$ 

```

```

      using linorder-not-less not-less-iff-gr-or-eq by fastforce
      then have (card (proj S W) * (1 + ε / (1 + ε))) / 2 ^ m <
card-slice h m
      apply (intro real-divide-pos-right)
      using * by auto
      then have h ∈ U m unfolding U-def
      by (simp add: μ-def)
    }
    ultimately have h ∈ L m ∨ h ∈ U m by blast
    then have ∀ m ∈ {Suc 0 .. < card S}.
      h ∈ T m →
      h ∈ T (m - Suc 0) ∨
      h ∉ L m ∧ h ∉ U m ⇒
      h ∈ T (card S - Suc 0) ⇒ False
      using m 1 by auto
  }
  ultimately show
    h ∈ (⋃ i ∈ {1 .. < card S}.
      (T i - T (i - 1)) ∩ (L i ∪ U i) ∪
      - T (card S - 1))
    by auto
qed

```

```

lemma smallest-nat-exists:
  assumes P i ¬P (0::nat)
  obtains m where m ≤ i P m ¬P (m-1)
  using assms
proof (induction i)
  case 0
  then show ?case by auto
next
  case (Suc i)
  then show ?case
    by (metis diff-Suc-1 le-Suc-eq)
qed

```

```

lemma mstar-non-zero:
  shows ¬ μ 0 * (1 + ε / (1 + ε)) ≤ thresh
proof -
  have μ 0 ≥ thresh
    unfolding μ-def
    by (auto simp add: thresh(2))
  thus ?thesis
    by (smt (verit, best) ε μ-gt-zero divide-pos-pos mult-le-cancel-left2)
qed

```

```

lemma real-div-less:
  assumes c > 0

```

**assumes**  $a \leq b * (c::nat)$   
**shows**  $real\ a / real\ c \leq b$   
**by** (*metis* *assms*(1) *assms*(2) *divide-le-eq* *of-nat-0-less-iff* *of-nat-mono* *of-nat-mult*)

**lemma** *mstar-exists*:

**obtains**  $m$  **where**

$\mu\ (m - 1) * (1 + \varepsilon / (1 + \varepsilon)) > thresh$

$\mu\ m * (1 + \varepsilon / (1 + \varepsilon)) \leq thresh$

$m \leq card\ S - 1$

**proof** –

**have**  $e1: 1 + \varepsilon / (1 + \varepsilon) > 1$

**by** (*simp* *add: \varepsilon* *add-nonneg-pos*)

**have**  $e2: 1 + \varepsilon / (1 + \varepsilon) < 2$

**by** (*simp* *add: \varepsilon* *add-nonneg-pos*)

**have**  $thresh \geq (4::real)$

**using** *thresh*(1) **by** *linarith*

**then have**  $up: thresh / (1 + \varepsilon / (1 + \varepsilon)) > 2$

**by** (*smt* (*verit*) *e1* *e2* *field-sum-of-halves* *frac-less2*)

**have**  $card\ (proj\ S\ W) \leq 2 * 2^{card\ S - Suc\ 0}$

**by** (*metis* *One-nat-def* *S* *Suc-diff-Suc* *card.empty* *card-gt-0-iff* *card-proj*(2) *diff-zero* *less-one* *order-less-le-trans* *plus-1-eq-Suc* *power-0* *power-Suc0-right* *power-add* *thresh*(1) *thresh*(2) *zero-neq-numeral*)

**then have**  $low: \mu\ (card\ S - 1) \leq 2$

**unfolding**  $\mu$ -*def*

**using** *real-div-less*

**by** (*smt* (*verit*) *Num.of-nat-simps*(2) *Num.of-nat-simps*(4) *Suc-1* *nat-zero-less-power-iff* *numeral-nat*(7) *of-nat-power* *plus-1-eq-Suc* *pos2*)

**have**  $pi: \mu\ (card\ S - 1) * (1 + \varepsilon / (1 + \varepsilon)) \leq thresh$

**using** *up* *low*

**by** (*smt* (*verit*, *ccfv-SIG*) *divide-le-eq* *e1*)

**from** *smallest-nat-exists*[*OF* *pi* *mstar-non-zero*]

**show** *?thesis*

**by** (*metis* *linorder-not-less* *that*)

**qed**

**definition** *mstar* :: *nat*

**where** *mstar* = (@*m*.

$\mu\ (m - 1) * (1 + \varepsilon / (1 + \varepsilon)) > thresh \wedge$

$\mu\ m * (1 + \varepsilon / (1 + \varepsilon)) \leq thresh \wedge$

$m \leq card\ S - 1$ )

**lemma** *mstar-prop*:

**shows**  
 $\mu (mstar - 1) * (1 + \varepsilon / (1 + \varepsilon)) > thresh$   
 $\mu mstar * (1 + \varepsilon / (1 + \varepsilon)) \leq thresh$   
 $mstar \leq card\ S - 1$   
**unfolding** *mstar-def*  
**by** (*smt (verit) some-eq-ex mstar-exists*)+

**lemma** *O1-lem*:  
**assumes**  $i \leq m$   
**shows**  $(T\ i - T\ (i-1)) \cap (L\ i \cup U\ i) \subseteq T\ m$   
**using** *T-mono assms* **by** *blast*

**lemma** *O1*:  
**shows**  $(\bigcup_{i \in \{1..mstar-3\}} (T\ i - T\ (i-1)) \cap (L\ i \cup U\ i)) \subseteq T\ (mstar-3)$   
**using** *T-mono* **by** *force*

**lemma** *T-anti-mono-neg*:  
**assumes**  $i \leq j$   
**shows**  $-T\ j \subseteq -T\ i$   
**by** (*simp add: Diff-mono T-mono assms*)

**lemma** *O2-lem*:  
**assumes**  $mstar < i$   
**shows**  $(T\ i - T\ (i-1)) \cap (L\ i \cup U\ i) \subseteq -T\ mstar$   
**proof** –  
**have**  $(T\ i - T\ (i-1)) \cap (L\ i \cup U\ i) \subseteq -T\ (i-1)$   
**by** *blast*  
**thus** *?thesis*  
**by** (*smt (verit) T-mono ApproxMCCore-axioms One-nat-def Suc-diff-Suc Suc-le-eq assms compl-mono diff-is-0-eq' diff-less-mono leI minus-nat.diff-0 subset-trans*)  
**qed**

**lemma** *O2*:  
**shows**  $(\bigcup_{i \in \{mstar..<card\ S\}} (T\ i - T\ (i-1)) \cap (L\ i \cup U\ i)) \cup -T\ (card\ S - 1) \subseteq L\ mstar \cup U\ mstar$   
**proof** –  
**have** *0*:  $(\bigcup_{i \in \{mstar..<card\ S\}} (T\ i - T\ (i-1)) \cap (L\ i \cup U\ i)) \subseteq (T\ mstar - T\ (mstar-1)) \cap (L\ mstar \cup U\ mstar) \cup (\bigcup_{i \in \{mstar+1..<card\ S\}} (T\ i - T\ (i-1)) \cap (L\ i \cup U\ i))$   
**apply** (*intro subsetI*)  
**apply** *clarsimp*  
**by** (*metis Suc-le-eq atLeastLessThan-iff basic-trans-rules(18)*)

**have** *1*:  $(\bigcup_{i \in \{mstar+1..<card\ S\}} (T\ i - T\ (i-1)) \cap (L\ i \cup U\ i))$

$(T\ i - T\ (i-1)) \cap (L\ i \cup U\ i) \subseteq -T\ mstar$   
**apply** *clarsimp*  
**by** (*metis* (*no-types*, *lifting*) *ApproxMCCore.T-mono* *ApproxMC-Core-axioms* *One-nat-def* *diff-Suc-1* *diff-le-mono* *subsetD*)

**have** 2:  $-T\ (card\ S - 1) \subseteq -T\ mstar$   
**using** *T-anti-mono-neg* *mstar-prop(3)* **by** *presburger*

**have** *thresh*  $\geq \mu\ mstar * (1 + \varepsilon / (1 + \varepsilon))$   
**using** *mstar-prop(2)* *thresh* **by** *linarith*

**then have** \*:  $-T\ mstar \subseteq U\ mstar$   
**unfolding** *T-def* *U-def*  
**by** *auto*

**have**  $(\bigcup_{i \in \{mstar..<card\ S\}} (T\ i - T\ (i-1)) \cap (L\ i \cup U\ i)) \cup -T\ (card\ S - 1) \subseteq ((T\ mstar - T\ (mstar-1)) \cap (L\ mstar \cup U\ mstar)) \cup -T\ mstar$   
**using** 0 1 2 **by** (*smt* (*z3*) *Un-iff* *subset-iff*)

**moreover have** ...  $\subseteq L\ mstar \cup U\ mstar$   
**using** \*  
**by** *blast*

**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** *O3*:

**assumes**  $i \leq mstar - 1$   
**shows**  $(T\ i - T\ (i-1)) \cap (L\ i \cup U\ i) \subseteq L\ i$

**proof** –

**have** \*:  $\mu\ i * (1 + \varepsilon / (1 + \varepsilon)) > thresh$

**by** (*smt* (*verit*, *ccfv-SIG*)  $\varepsilon$   *$\mu$ -anti-mono* *assms* *divide-nonneg-nonneg* *mstar-prop(1)* *mult-right-mono*)

**have**  $x \in T\ i \wedge x \in U\ i \implies False$  **for**  $x$   
**unfolding** *T-def* *U-def*

**using** \* **by** *auto*

**thus** *?thesis*

**by** *blast*

**qed**

**lemma** *union-split-lem*:

**assumes**  $x \in (\bigcup_{i \in \{1..<n::nat\}} P\ i)$

**shows**  $x \in (\bigcup_{i \in \{1..m-3\}} P\ i) \cup$

$P\ (m-2) \cup$

$P\ (m-1) \cup$

$(\bigcup_{i \in \{m..<n\}} P\ i)$

**proof** –  
**obtain**  $i$  **where**  $i: i \in \{1..<n\}$   
 $x \in P\ i$  **using**  $x$  **by** *auto*  
**have**  $i \in \{1..m-3\} \vee i = m-2 \vee i = m-1 \vee i \in \{m..<n\}$   
**using**  $i(1)$   
**by** *auto*  
**thus** *?thesis* **using**  $i$   
**by** *blast*  
**qed**

**lemma** *union-split*:  
 $(\bigcup_{i \in \{1..<n::nat\}}. P\ i) \subseteq$   
 $(\bigcup_{i \in \{1..m-3\}}. P\ i) \cup$   
 $P\ (m-2) \cup$   
 $P\ (m-1) \cup$   
 $(\bigcup_{i \in \{m..<n\}}. P\ i)$   
**using** *union-split-lem*  
**by** (*metis subsetI*)

**lemma** *failure-bound*:  
**shows** *approxcore-fail*  $\subseteq$   
 $T\ (mstar-3) \cup$   
 $L\ (mstar-2) \cup$   
 $L\ (mstar-1) \cup$   
 $(L\ mstar \cup U\ mstar)$   
**proof** –

**have**  $*$ : *approxcore-fail*  $\subseteq$   
 $(\bigcup_{i \in \{1..<card\ S\}}. (T\ i - T\ (i-1)) \cap (L\ i \cup U\ i)) \cup -T\ (card\ S - 1)$  (**is** -  $\subseteq$   
 $(\bigcup_{i \in \{1..<card\ S\}}. ?P\ i) \cup -)$   
**using** *failure-imp* .

**moreover have** ...  $\subseteq$   
 $(\bigcup_{i \in \{1..mstar-3\}}. ?P\ i) \cup$   
 $?P\ (mstar-2) \cup$   
 $?P\ (mstar-1) \cup$   
 $((\bigcup_{i \in \{mstar..<card\ S\}}. ?P\ i) \cup -T\ (card\ S - 1))$   
**using**  
*union-split*[of  $\lambda i. ?P\ i\ card\ S\ mstar$ ]  
**by** *blast*

**moreover have** ...  $\subseteq$   
 $T\ (mstar-3) \cup$   
 $L\ (mstar-2) \cup$   
 $L\ (mstar-1) \cup$   
 $(L\ mstar \cup U\ mstar)$

**using** *O1 O2 O3*  
**by** (*metis (no-types, lifting) One-nat-def Un-mono diff-Suc-eq-diff-pred*)

```

diff-le-self nat-1-add-1 plus-1-eq-Suc)
  ultimately show ?thesis
    by (meson order-trans)
qed

end

end

```

## 5 ApproxMCCore analysis

This section analyzes ApproxMCCore with respect to a universal hash family. The proof follows Lemmas 1 and 2 from Chakraborty et al. [1].

```

theory ApproxMCCoreAnalysis imports
  ApproxMCCore
begin

```

```

definition Hslice :: nat ⇒
  ('a assg ⇒ 'b ⇒ nat assg) ⇒ ('a assg ⇒ 'b ⇒ nat assg)
  where Hslice i H = (λw s. aslice i (H w s))

```

```

context prob-space
begin

```

```

lemma indep-vars-prefix:
  assumes indep-vars (λ-. count-space UNIV) H J
  shows indep-vars (λ-. count-space UNIV) (Hslice i H) J
proof -
  have indep-vars (λ-. count-space UNIV) (λy. (λx. aslice i x) ◦ H y)
  J
  by (auto intro!: indep-vars-compose[OF assms(1)])
  thus ?thesis
    unfolding o-def Hslice-def
    by auto
qed

```

```

lemma assg-nonempty-dom:
  shows
    (λx. if x < i then Some True else None) ∈
      {α::nat assg. dom α = {0..} }
  by (auto split: if-splits)

```

```

lemma card-dom-ran-nat-assg:
  shows card {α::nat assg. dom α = {0..} } = 2n
proof -
  have *: {α::nat assg. dom α = {0..} } =

```

```

    {w. dom w = {0..<n} ∧ ran w ⊆ {True,False}}
  by auto
  have finite {0..<n} by auto
  from card-dom-ran[OF this]
  have card {w. dom w = {0..<n} ∧ ran w ⊆ {True,False}} =
    card {True,False} ^ card {0..<n} .
  thus ?thesis
    unfolding *
    by (auto simp add: numeral-2-eq-2)
qed

```

```

lemma card-nat-assg-le:
  assumes i ≤ n
  shows card {α::nat assg. dom α = {0..<n}} =
    2n-i * card {α::nat assg. dom α = {0..<i}}
  unfolding card-dom-ran-nat-assg
  by (metis assms le-add-diff-inverse mult.commute power-add)

```

```

lemma empty-nat-assg-slice-notin:
  assumes i ≤ n
  assumes dom β ≠ {0..<i}
  shows {α::nat assg. dom α = {0..<n} ∧ aslice i α = β} = {}
proof (intro equals0I)
  fix x
  assume x ∈ {α. dom α = {0..<n} ∧ aslice i α = β}
  then have dom β = {0..<i}
    unfolding aslice-def
    using assms(1) by force
  thus False using assms(2) by blast
qed

```

```

lemma restrict-map-dom:
  shows α |' dom α = α
  unfolding restrict-map-def fun-eq-iff
  by (simp add: domIff)

```

```

lemma aslice-refl:
  assumes dom α = {..<i}
  shows aslice i α = α
  unfolding aslice-def assms[symmetric]
  using restrict-map-dom by auto

```

```

lemma bij-betw-with-inverse:
  assumes f ' A ⊆ B
  assumes ∧x. x ∈ A ⇒ g (f x) = x
  assumes g ' B ⊆ A
  assumes ∧x. x ∈ B ⇒ f (g x) = x
  shows bij-betw f A B
proof -

```



```

have inj-on f A
  by (metis assms(2) inj-onI)

thus ?thesis
  unfolding bij-betw-def
  using assms
  by (metis image-subset-iff subsetI subset-antisym)
qed

lemma card-nat-assg-slice:
  assumes  $i \leq n$ 
  assumes  $\text{dom } \beta = \{0..<i\}$ 
  shows  $\text{card } \{\alpha::\text{nat assg. } \text{dom } \alpha = \{0..<n\} \wedge \text{aslice } i \alpha = \beta\} =$ 
     $2^{i} \wedge^{(n-i)}$ 
proof -
  have  $\text{dom } \alpha = \{0..<i\} \wedge \text{aslice } i \alpha = \beta \longleftrightarrow \alpha = \beta$  for  $\alpha$ 
    using aslice-refl
    by (metis assms(2) lessThan-atLeast0)
  then have r2:
     $\{\alpha::\text{nat assg. } \text{dom } \alpha = \{0..<i\} \wedge \text{aslice } i \alpha = \beta\} = \{\beta\}$ 
    by simp

define f where f =
  ( $\lambda(\alpha::\text{nat assg}, \beta::\text{nat assg}) j.$ 
     $\text{if } j < i \text{ then } \beta j \text{ else } \alpha (j-i)$ )
let ?lhs = ( $\{\alpha. \text{dom } \alpha = \{0..<n-i\}\} \times$ 
   $\{\alpha. \text{dom } \alpha = \{0..<i\} \wedge \text{aslice } i \alpha = \beta\}$ )
let ?rhs =  $\{\alpha::\text{nat assg. } \text{dom } \alpha = \{0..<n\} \wedge \text{aslice } i \alpha = \beta\}$ 

define finv where finv =
  ( $\lambda fab::\text{nat assg.}$ 
    ( $\lambda j. fab (j+i),$ 
       $fab \upharpoonright \{..<i\}$ )
  )
have 11:  $\bigwedge x. \text{dom } (f x) = \{j. \text{if } j < i \text{ then } j \in \text{dom } (\text{snd } x) \text{ else } j$ 
   $- i \in \text{dom } (\text{fst } x)\}$ 
unfolding f-def
  by (auto simp add: fun-eq-iff split: if-splits)
have 1:  $f \upharpoonright \{..<i\} \subseteq ?rhs$ 
proof standard
  fix x
  assume  $x: x \in f \upharpoonright \{..<i\} (\{\alpha. \text{dom } \alpha = \{0..<n-i\}\} \times \{\alpha. \text{dom } \alpha =$ 
 $\{0..<i\} \wedge \text{aslice } i \alpha = \beta\})$ 
  obtain a b where  $ab: \text{dom } a = \{0..<n-i\} \text{ dom } b = \{0..<i\}$ 
   $x = f (a,b)$ 
  using x by blast
  have 1:  $\text{dom } x = \{0..<n\}$ 
  unfolding ab 11 using assms(1)
  by (auto simp add: ab)

```

**have** 2:  $aslice\ i\ x = \beta$   
**using**  $x$  **unfolding**  $f$ -def  
**by** (*auto simp add: aslice-def restrict-map-def split: if-splits*)  
**show**  $x \in \{\alpha. dom\ \alpha = \{0..<n\} \wedge aslice\ i\ \alpha = \beta\}$   
**using** 1 2 **by** *blast*  
**qed**

**have**  $dom\ a = \{0..<n - i\} \implies$   
 $dom\ b = \{0..<i\} \implies$   
 $\forall x. aslice\ i\ b\ x = \beta\ x \implies$   
 $\neg x < i \implies None = b\ x$  **for**  $a\ b :: nat\ assg$  **and**  $x$   
**by** (*metis atLeastLessThan-iff domIff*)  
**then have** 2:  $\bigwedge x. x \in ?lhs \implies finv\ (f\ x) = x$   
**unfolding**  $finv$ -def  $f$ -def  
**by** (*clarsimp simp add: fun-eq-iff restrict-map-def*)

**have** 31:  $\bigwedge fab\ x\ y.$   
 $dom\ fab = \{0..<n\} \implies$   
 $\beta = aslice\ i\ fab \implies$   
 $fab\ (x + i) = Some\ y \implies$   
 $x < n - i$   
**by** (*metis atLeastLessThan-iff domI less-diff-conv*)

**also have**  $\bigwedge fab\ x.$   
 $dom\ fab = \{0..<n\} \implies$   
 $\beta = aslice\ i\ fab \implies$   
 $x < i \implies \exists y. fab\ x = Some\ y$   
**by** (*metis assms(1) domD dual-order.trans lessThan-atLeast0 lessThan-iff linorder-not-less*)

**ultimately have** 3:  $finv\ ' ?rhs \subseteq ?lhs$   
**unfolding**  $finv$ -def  
**by** (*auto simp add: aslice-def split: if-splits*)

**have** 4:  $\bigwedge x. x \in ?rhs \implies f\ (finv\ x) = x$   
**unfolding**  $finv$ -def  $f$ -def  
**by** (*auto simp add: fun-eq-iff restrict-map-def*)

**have** *bij-betw*  $f\ ?lhs\ ?rhs$   
**by** (*auto intro: bij-betw-with-inverse[OF 1 2 3 4]*)

**from** *bij-betw-same-card*[*OF this*]

**have**  
 $card\ \{\alpha :: nat\ assg. dom\ \alpha = \{0..<n\} \wedge aslice\ i\ \alpha = \beta\} =$   
 $card\ (\{\alpha :: nat\ assg. dom\ \alpha = \{0..<n-i\}\} \times$   
 $\{\alpha :: nat\ assg. dom\ \alpha = \{0..<i\} \wedge aslice\ i\ \alpha = \beta\})$   
**by** *auto*

**moreover have**  $\dots = 2^{(n-i)}$   
**using**  $r2$  *card-dom-ran-nat-assg*  
**by** (*simp add: card-cartesian-product*)

**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** *finite-dom*:  
**assumes** *finite* *V*  
**shows** *finite*  $\{w :: 'a \rightarrow \text{bool}. \text{dom } w = V\}$   
**proof** –  
**have** \*:  $\{w :: 'a \rightarrow \text{bool}. \text{dom } w = V\} =$   
 $\{w. \text{dom } w = V \wedge \text{ran } w \subseteq \{\text{True}, \text{False}\}\}$   
**by** *auto*  
**show** *?thesis* **unfolding** \*  
**apply** (*intro finite-set-of-finite-maps*)  
**using** *assms* **by** *auto*  
**qed**

**lemma** *universal-prefix-family-from-hash*:  
**assumes** *M*:  $M = \text{measure-pmf } p$   
**assumes** *kH*: *k-universal* *k* *H* *D*  $\{\alpha :: \text{nat assg}. \text{dom } \alpha = \{0..<n\}\}$   
**assumes** *i*:  $i \leq n$   
**shows** *k-universal* *k* (*Hslice* *i* *H*) *D*  $\{\alpha. \text{dom } \alpha = \{0..<i\}\}$   
**proof** –  
**have** *k-wise-indep-vars* *k* ( $\lambda-. \text{count-space UNIV}$ ) *H* *D*  
**using** *kH* **unfolding** *k-universal-def*  
**by** *auto*  
**then have** 1: *k-wise-indep-vars* *k* ( $\lambda-. \text{count-space UNIV}$ ) (*Hslice* *i*  
*H*) *D*  
**unfolding** *k-wise-indep-vars-def*  
**using** *indep-vars-prefix*  
**by** *auto*

**have** *fdom*: *finite*  $\{\alpha :: \text{nat assg}. \text{dom } \alpha = \{0..<i\}\}$   
**using** *measure-pmf.finite-dom* **by** *blast*  
**have** *nempty*:  $\{\alpha :: \text{nat assg}. \text{dom } \alpha = \{0..<i\}\} \neq \{\}$   
**using** *assg-nonempty-dom*  
**by** (*metis empty-iff*)  
**have** 2:  $x \in D \implies$   
*uniform-on* (*Hslice* *i* *H* *x*)  $\{\alpha. \text{dom } \alpha = \{0..<i\}\}$  **for** *x*  
**proof** –  
**assume**  $x \in D$   
**then have** *unif*: *uniform-on* (*H* *x*)  $\{\alpha. \text{dom } \alpha = \{0..<n\}\}$   
**by** (*metis kH k-universal-def*)  
**show** *uniform-on* (*Hslice* *i* *H* *x*)  $\{\alpha. \text{dom } \alpha = \{0..<i\}\}$   
**proof** (*intro uniform-onI[OF M fdom nempty]*)  
**fix**  $\beta$   
**have** \*:  $\{\omega \in \text{space } M. H \ x \ \omega \in \{\alpha. \text{aslice } i \ \alpha = \beta\}\} =$   
 $\{\omega. \text{Hslice } i \ H \ x \ \omega = \beta\}$   
**unfolding** *Hslice-def*  
**by** (*auto simp add: M*)

```

have { $\alpha::\text{nat assg. dom } \alpha = \{0..<n\}\} \cap \{\alpha. \text{ aslice } i \alpha = \beta\} =$ 
  { $\alpha::\text{nat assg. dom } \alpha = \{0..<n\} \wedge \text{ aslice } i \alpha = \beta\}$ 
by auto

then have (card ({ $\alpha::\text{nat assg. dom } \alpha = \{0..<n\}\} \cap \{\alpha. \text{ aslice } i \alpha = \beta\}$ ))
  = (if dom  $\beta = \{0..<i\}$  then  $2^{(n-i)}$  else 0)
using card-nat-assg-slice[OF i]
by (simp add: empty-nat-assg-slice-notin i)

then have **: real (card ({ $\alpha::\text{nat assg. dom } \alpha = \{0..<n\}\} \cap \{\alpha. \text{ aslice } i \alpha = \beta\}$ ))
  = indicat-real { $\alpha. \text{ dom } \alpha = \{0..<i\}\} \beta * 2^{(n-i)}$ 
by simp

from uniform-onD[OF unif, of { $\alpha. \text{ aslice } i \alpha = \beta\}$ ]
have prob { $\omega. \text{ Hslice } i H x \omega = \beta\} =$ 
  real (card ({ $\alpha. \text{ dom } \alpha = \{0..<n\}\} \cap \{\alpha. \text{ aslice } i \alpha = \beta\}$ )) /
  real (card { $\alpha::\text{nat assg. dom } \alpha = \{0..<n\}\}$ )
unfolding * by auto
moreover have ... =
  indicat-real { $\alpha. \text{ dom } \alpha = \{0..<i\}\} \beta * 2^{(n-i)}$  /
  real (card { $\alpha::\text{nat assg. dom } \alpha = \{0..<n\}\}$ )
unfolding ** by auto
moreover have ... = indicat-real { $\alpha. \text{ dom } \alpha = \{0..<i\}\} \beta * 2^{(n-i)}$  /  $2^n$ 
by (simp add: card-dom-ran-nat-assg)
moreover have ... = indicat-real { $\alpha. \text{ dom } \alpha = \{0..<i\}\} \beta / 2^i$ 
by (simp add: i power-diff)
moreover have ... = indicat-real { $\alpha. \text{ dom } \alpha = \{0..<i\}\} \beta /$ 
  real (( $2^i::\text{nat}$ )) by auto
moreover have ... = indicat-real { $\alpha. \text{ dom } \alpha = \{0..<i\}\} \beta /$ 
  real (card { $\alpha::\text{nat assg. dom } \alpha = \{0..<i\}\}$ )
using card-dom-ran-nat-assg
by auto
ultimately show prob { $\omega. \text{ Hslice } i H x \omega = \beta\} =$ 
  indicat-real { $\alpha. \text{ dom } \alpha = \{0..<i\}\} \beta /$ 
  real (card { $\alpha::\text{nat assg. dom } \alpha = \{0..<i\}\}$ )
by presburger
qed
qed
show ?thesis
unfolding k-universal-def
using 1 2
by blast
qed
end

```

```

context ApproxMCCore
begin

definition pivot :: real
  where pivot = 9.84 * ( 1 + 1 / ε ) ^ 2

context
  assumes thresh: thresh ≥ ( 1 + ε / ( 1 + ε ) ) * pivot
begin

lemma aux-1:
  assumes fin:finite (set-pmf p)
  assumes σ: σ > 0
  assumes exp: μ i = measure-pmf.expectation p Y
  assumes var: σ ^ 2 = measure-pmf.variance p Y
  assumes var-bound: σ ^ 2 ≤ μ i
  shows
    measure-pmf.prob p {y. | Y y - μ i | ≥ ε / ( 1 + ε ) * μ i }
    ≤ ( 1 + ε ) ^ 2 / ( ε ^ 2 * μ i )
proof -
  have pvar: measure-pmf.variance p Y > 0
  using var σ
  by (metis zero-less-power)
  have kmu: ε * (μ i) / ((1 + ε) * σ) > 0
  using σ pvar var var-bound
  using ε by auto
  have mupos: μ i > 0
  using pvar var var-bound by linarith
  from spec-chebyshev-inequality [OF fin pvar kmu]
  have measure-pmf.prob p
    {y. (Y y - measure-pmf.expectation p Y) ^ 2 ≥
      (ε * (μ i) / ((1 + ε) * σ)) ^ 2 * measure-pmf.variance p Y} ≤
    1 / (ε * (μ i) / ((1 + ε) * σ)) ^ 2
  by simp
  then have ineq1:measure-pmf.prob p
    {y. (Y y - μ i) ^ 2 ≥
      (ε * (μ i) / ((1 + ε) * σ)) ^ 2 * σ ^ 2} ≤ 1 / (ε * (μ i) / ((1 +
    ε) * σ)) ^ 2
  using exp var by simp
  have (λy. (Y y - μ i) ^ 2 ≥ (ε * (μ i) / ((1 + ε) * σ)) ^ 2 * σ ^ 2)
    = (λy. (Y y - μ i) ^ 2 ≥ (ε * (μ i) / ((1 + ε) * σ) * σ) ^ 2)
  by (metis power-mult-distrib)
  moreover have ... = (λy. | Y y - μ i | ≥ ε * (μ i) / ((1 + ε) * σ)
  * σ)
proof -
  have 0 ≤ ε * μ i / ( 1 + ε )
  by (simp add: ε less-eq-real-def mupos zero-compare-simps(1))
  then show ?thesis

```

**apply** *clarsimp*  
**by** (*metis abs-le-square-iff abs-of-nonneg*)  
**qed**  
**moreover have** ... =  $(\lambda y. | Y y - \mu i | \geq \varepsilon * (\mu i) / (1 + \varepsilon))$   
**using**  $\sigma$  **by** *auto*  
**ultimately have** *simp1*:  $(\lambda y. (Y y - \mu i)^2 \geq (\varepsilon * (\mu i) / ((1 + \varepsilon) * \sigma))^2 * \sigma^2)$   
=  $(\lambda y. | Y y - \mu i | \geq \varepsilon / (1 + \varepsilon) * (\mu i))$   
**by** *auto*  
  
**have**  $\sigma^2 / (\mu i)^2 \leq (\mu i) / (\mu i)^2$   
**using** *var-bound*  *$\mu$ -gt-zero*  
**by** (*simp add: divide-right-mono*)  
**moreover have** ... =  $1 / (\mu i)$   
**by** (*simp add: power2-eq-square*)  
**ultimately have** *simp2*:  $\sigma^2 / (\mu i)^2 \leq 1 / (\mu i)$  **by** *auto*  
  
**have** *measure-pmf.prob*  $p \{y. | Y y - \mu i | \geq \varepsilon / (1 + \varepsilon) * (\mu i)\}$   
 $\leq 1 / (\varepsilon * (\mu i) / ((1 + \varepsilon) * \sigma))^2$   
**using** *ineq1 simp1*  
**by** *auto*  
**moreover have** ... =  $(1 + \varepsilon)^2 / \varepsilon^2 * \sigma^2 / (\mu i)^2$   
**by** (*simp add: power-divide power-mult-distrib*)  
**moreover have** ...  $\leq (1 + \varepsilon)^2 / \varepsilon^2 * (1 / (\mu i))$   
**using** *simp2*  
**by** (*smt (verit, best)  $\varepsilon$  divide-pos-pos mult-left-mono times-divide-eq-right zero-less-power*)  
**ultimately have** *measure-pmf.prob*  $p \{y. | Y y - \mu i | \geq \varepsilon / (1 + \varepsilon) * (\mu i)\}$   
 $\leq (1 + \varepsilon)^2 / (\varepsilon^2 * (\mu i))$   
**by** *auto*  
**thus** *?thesis* **by** *auto*  
**qed**

**lemma** *analysis-1-1*:  
**assumes** *p*: *finite (set-pmf p)*  
**assumes** *ind*: *prob-space.k-universal (measure-pmf p) 2 H*  
 $\{\alpha::'a \text{ assg. } \text{dom } \alpha = S\}$   
 $\{\alpha::\text{nat assg. } \text{dom } \alpha = \{0..<\text{card } S - 1\}\}$   
**assumes** *i*:  $i \leq \text{card } S - 1$   
**shows**  
*measure-pmf.prob*  $p$   
 $\{s. | \text{card-slice } ((\lambda w. H w s)) i - \mu i | \geq \varepsilon / (1 + \varepsilon) * \mu i\}$   
 $\leq (1 + \varepsilon)^2 / (\varepsilon^2 * \mu i)$   
**proof** –

**define** *var* **where** *var* =

```

      (λi. measure-pmf.variance p
        (λs. real (card (proj S W ∩
          {w. Hslice i H w s = aslice i α}))))))

from prob-space.universal-prefix-family-from-hash[OF - - ind]
have hf: prob-space.k-universal (measure-pmf p) 2
  (Hslice i H) {α::'a assg. dom α = S} {α. dom α = {0..<i}}
  using prob-space-measure-pmf
  using i ind prob-space.universal-prefix-family-from-hash
  by blast

have pSW: proj S W ⊆ {α. dom α = S}
  unfolding proj-def restr-def by (auto split:if-splits)

have ain: aslice i α ∈ {α. dom α = {0..<i}} using α
  unfolding aslice-def using i by auto

from k-universal-expectation-eq[OF p hf finite-proj-S pSW ain]
have exp:measure-pmf.expectation p
  (λs. real (card (proj S W ∩
    {w. Hslice i H w s = aslice i α}))) =
  real (card (proj S W)) /
  real (card {α::nat assg. dom α = {0..<i}}) .

have exp-mu:real (card (proj S W)) /
  real (card {α::nat assg. dom α = {0..<i}}) = μ i
  unfolding μ-def
  by (simp add: measure-pmf.card-dom-ran-nat-assg)

have proj S W ∩
  {w. Hslice i H w s = aslice i α} =
  proj S W ∩ {w. hslice i (λw. H w s) w = aslice i α} for s
  unfolding proj-def Hslice-def hslice-def
  by auto
then have extend-card-slice:
  (λs. (card (proj S W ∩ {w. Hslice i H w s = aslice i α}))) =
  card-slice ((λw. H w s) i)
  unfolding card-slice-def by auto

have mu-exp: μ i = measure-pmf.expectation p
  (λs. real (card (proj S W ∩ {w. Hslice i H w s = aslice i α})))
  using exp exp-mu by auto

from two-universal-variance-bound[OF p hf finite-proj-S pSW ain]
have
  var i ≤
  measure-pmf.expectation p
  (λs. real (card (proj S W ∩
    {w. Hslice i H w s = aslice i α})))

```

**unfolding** *var-def* .  
**then have** *var-bound*:  $\text{var } i \leq \mu i$  **using** *exp exp-mu*  
**by** *linarith*

**have**  $\text{var } i \geq 0$  **unfolding** *var-def* **by** *auto*  
**then have**  $\text{var } i > 0 \vee \text{var } i = 0$  **by** *auto*  
**moreover** {  
**assume**  $\text{var } i = 0$

**then have** 1:  
*measure-pmf.expectation p*  
 $(\lambda s. (\text{card-slice } (\lambda w. H w s) i - \mu i)^{\wedge 2}) = 0$   
**unfolding** *var-def extend-card-slice mu-exp* **by** *auto*

**have** 2: *measure-pmf.expectation p*  
 $(\lambda s. (\text{card-slice } (\lambda w. H w s) i - \mu i)^{\wedge 2}) =$   
 $\text{sum } (\lambda s. (\text{card-slice } (\lambda w. H w s) i - \mu i)^{\wedge 2} * \text{pmf } p s) (\text{set-pmf}$   
*p)*  
**using** *assms* **by** (*auto intro!*: *integral-measure-pmf-real*)

**have**  $\forall x \in \text{set-pmf } p. (\text{card-slice } (\lambda w. H w x) i - \mu i)^{\wedge 2} * \text{pmf } p$   
 $x = 0$   
**apply** (*subst sum-nonneg-eq-0-iff[symmetric]*)  
**using** 1 2 *p* **by** *auto*  
**then have** \*:  $\bigwedge x. x \in \text{set-pmf } p \implies (\text{card-slice } (\lambda w. H w x) i -$   
 $\mu i)^{\wedge 2} = 0$   
**by** (*meson mult-eq-0-iff set-pmf-iff*)

**have** \*\*:  $(1 + \varepsilon)^{\wedge 2} / (\varepsilon^{\wedge 2} * \mu i) > 0$   
**using** *mu-gt-zero*  $\varepsilon$  **by** *auto*  
**have**  $\varepsilon / (1 + \varepsilon) * \mu i > 0$   
**using** *mu-gt-zero*  $\varepsilon$  **by** *auto*  
**then have**  $\bigwedge s. s \in \text{set-pmf } p \implies | \text{card-slice } ((\lambda w. H w s)) i - \mu$   
 $i | < \varepsilon / (1 + \varepsilon) * \mu i$   
**using** \* **by** *auto*  
**then have**  
*measure-pmf.prob p*  
 $\{s. | \text{card-slice } ((\lambda w. H w s)) i - \mu i | \geq \varepsilon / (1 + \varepsilon) * \mu i\} = 0$   
**apply** (*subst measure-pmf-zero-iff*)  
**using** *linorder-not-less* **by** *auto*  
**then have**  
*measure-pmf.prob p*  
 $\{s. | \text{card-slice } ((\lambda w. H w s)) i - \mu i | \geq \varepsilon / (1 + \varepsilon) * \mu i\}$   
 $\leq (1 + \varepsilon)^{\wedge 2} / (\varepsilon^{\wedge 2} * \mu i)$   
**using** \*\* **by** *auto*

**}**  
**moreover** {  
**define** *sigma* **where**  $\text{sigma} = \text{sqrt}(\text{var } i)$   
**assume**  $\text{var } i > 0$



**then have** *sigma-gt-0*:  $\sigma > 0$   
**unfolding** *sigma-def* **by** *simp*

**have** *extend-sigma*:  $\sigma^2 = \text{measure-pmf.variance } p$   
 $(\lambda s. \text{real } (\text{card } (\text{proj } S \ W \ \cap \ \{w. \text{Hslice } i \ H \ w \ s = \text{aslice } i \ \alpha\})))$   
**unfolding** *sigma-def* *var-def*  
**using** *less-eq-real-def* *local.var-def* *real-sqrt-pow2* *sigma-def* *sigma-gt-0*  
**by** *fastforce*

**have** *sigma-bound*:  $\sigma^2 \leq \mu \ i$   
**using** *var-bound* *sigma-gt-0*  
**using** *sigma-def* **by** *force*

**from** *aux-1*[*OF* *p* *sigma-gt-0* *mu-exp* *extend-sigma* *sigma-bound*]  
**have**  
*measure-pmf.prob* *p*  
 $\{s. \mid \text{card-slice } ((\lambda w. \text{H } w \ s)) \ i - \mu \ i \mid \geq \varepsilon / (1 + \varepsilon) * \mu \ i \}$   
 $\leq (1 + \varepsilon)^2 / (\varepsilon^2 * \mu \ i)$   
**using** *extend-card-slice* **by** *auto*

**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** *analysis-1-2*:  
**assumes** *p*: *finite* (*set-pmf* *p*)  
**assumes** *ind*: *prob-space.k-universal* (*measure-pmf* *p*) 2 *H*  
 $\{\alpha::'a \text{ assg. } \text{dom } \alpha = S\}$   
 $\{\alpha::\text{nat assg. } \text{dom } \alpha = \{0..<\text{card } S - 1\}\}$   
**assumes** *i*:  $i \leq \text{card } S - 1$   
**assumes**  $\beta: \beta \leq 1$   
**shows** *measure-pmf.prob* *p*  
 $\{s. \text{real}(\text{card-slice } ((\lambda w. \text{H } w \ s)) \ i) \leq \beta * \mu \ i \}$   
 $\leq 1 / (1 + (1 - \beta)^2 * \mu \ i)$

**proof** –  
**have**  $*$ :  $(\bigwedge s. (0::\text{real}) \leq \text{card-slice } ((\lambda w. \text{H } w \ s)) \ i)$   
**by** *simp*  
**from** *spec-paley-zygmund-inequality*[*OF*  $p * \beta$ ]  
**have** *paley-zygmund*:  
 $(\text{measure-pmf.variance } p \ (\lambda s. \text{real}(\text{card-slice } ((\lambda w. \text{H } w \ s)) \ i))$   
 $+ (1 - \beta)^2 * (\text{measure-pmf.expectation } p \ (\lambda s. \text{card-slice } ((\lambda w. \text{H } w \ s)) \ i))^2) *$   
 $\text{measure-pmf.prob } p \ \{s. \text{real}(\text{card-slice } ((\lambda w. \text{H } w \ s)) \ i) > \beta * \text{measure-pmf.expectation } p \ (\lambda s. \text{real}(\text{card-slice } ((\lambda w. \text{H } w \ s)) \ i))\} \geq$   
 $(1 - \beta)^2 * (\text{measure-pmf.expectation } p \ (\lambda s. \text{real}(\text{card-slice } ((\lambda w. \text{H } w \ s)) \ i)))^2$

```

by auto
define var where var = ( $\lambda i$ . measure-pmf.variance p
  ( $\lambda s$ . real (card (proj S W  $\cap$ 
    {w. Hslice i H w s = aslice i  $\alpha$ }))))

from prob-space.universal-prefix-family-from-hash[OF - - ind]
have hf: prob-space.k-universal (measure-pmf p) 2
  (Hslice i H) { $\alpha$ ::'a assg. dom  $\alpha$  = S} { $\alpha$ . dom  $\alpha$  = {0..i}}
  using prob-space-measure-pmf
  using i ind prob-space.universal-prefix-family-from-hash
  by blast

have pSW: proj S W  $\subseteq$  { $\alpha$ . dom  $\alpha$  = S}
  unfolding proj-def restr-def
  by (auto split:if-splits)

have ain: aslice i  $\alpha \in$  { $\alpha$ . dom  $\alpha$  = {0..i}} using  $\alpha$ 
  unfolding aslice-def using i by auto

from k-universal-expectation-eq[OF p hf finite-proj-S pSW ain]
have exp:measure-pmf.expectation p
  ( $\lambda s$ . real (card (proj S W  $\cap$ 
    {w. Hslice i H w s = aslice i  $\alpha$ }))) =
  real (card (proj S W)) /
  real (card { $\alpha$ ::nat assg. dom  $\alpha$  = {0..i}}) .

have exp-mu:real (card (proj S W)) /
  real (card { $\alpha$ ::nat assg. dom  $\alpha$  = {0..i}}) =  $\mu$  i
  unfolding  $\mu$ -def
  by (simp add: measure-pmf.card-dom-ran-nat-assg)

have proj S W  $\cap$ 
  {w. Hslice i H w s = aslice i  $\alpha$ } =
  proj S W  $\cap$  {w. hslice i ( $\lambda w$ . H w s) w = aslice i  $\alpha$ } for s
  unfolding proj-def Hslice-def hslice-def
  by auto

then have extend-card-slice: $\wedge s$ . (card (proj S W  $\cap$ 
  {w. Hslice i H w s = aslice i  $\alpha$ })) =
  card-slice (( $\lambda w$ . H w s)) i
  unfolding card-slice-def by auto

have mu-exp:  $\mu$  i = measure-pmf.expectation p
  ( $\lambda s$ . real (card (proj S W  $\cap$ 
    {w. Hslice i H w s = aslice i  $\alpha$ })))
  using exp exp-mu by auto

from two-universal-variance-bound[OF p hf finite-proj-S pSW ain]
have

```

```

var i ≤
measure-pmf.expectation p
(λs. real (card (proj S W ∩
{w. Hslice i H w s = aslice i α})))
unfolding var-def .
then have var-bound: var i ≤ μ i using exp exp-mu
by linarith

have pos-mu: μ i > 0
unfolding μ-def
by (simp add: card-proj-witnesses)

have comp: measure-pmf.prob p
{s. β * μ i < real (card-slice (λw. H w s) i)} =
(1 - measure-pmf.prob p {s. real(card-slice ((λw. H w s)) i) ≤ β
* μ i})
apply (subst measure-pmf.prob-compl[symmetric])
by (auto intro!: arg-cong[where f = measure-pmf.prob p])

have extend-var-bound: measure-pmf.variance p (λs. card-slice ((λw.
H w s)) i) ≤ μ i
using var-bound
unfolding var-def
by (simp add: extend-card-slice)
then have
(measure-pmf.variance p (λs. real(card-slice ((λw. H w s)) i))
+ (1-β)2 * (measure-pmf.expectation p (λs. real(card-slice
((λw. H w s)) i) ) )2)
* measure-pmf.prob p {s. real(card-slice ((λw. H w s)) i) > β *
measure-pmf.expectation p (λs. real(card-slice ((λw. H w s)) i) )} ≤
(μ i + (1-β)2 * (measure-pmf.expectation p ( λs. real(card-slice
((λw. H w s)) i) ) )2)
* measure-pmf.prob p
{s. real(card-slice ((λw. H w s)) i) > β * measure-pmf.expectation
p (λs. real(card-slice ((λw. H w s)) i) )}
by (auto intro!: mult-right-mono)
moreover have
... ≤ (μ i + (1-β)2 * (measure-pmf.expectation p ( λs. real(card-slice
((λw. H w s)) i) ) )2)
* measure-pmf.prob p
{s. real(card-slice ((λw. H w s)) i) > β * measure-pmf.expectation
p (λs. real(card-slice ((λw. H w s)) i) )}
by fastforce
ultimately have
(μ i + (1-β)2 * (μ i)2) * measure-pmf.prob p {s. real(card-slice
((λw. H w s)) i) > β * μ i} ≥ (1-β)2 * (μ i)2
unfolding mu-exp extend-card-slice
using paley-zigmund by linarith
then have

```

$(1 + (1-\beta)^2 * (\mu i)) * (\mu i) * \text{measure-pmf.prob } p \{s. \text{real}(\text{card-slice } ((\lambda w. H w s) i) > \beta * \mu i)\} \geq (1-\beta)^2 * (\mu i) * (\mu i)$   
**by** (*smt* (*verit*) *left-diff-distrib'* *more-arith-simps*(11) *mult-cancel-right mult-cancel-right2 power2-eq-square*)  
**then have**  
 $(1 + (1-\beta)^2 * (\mu i)) * \text{measure-pmf.prob } p \{s. \text{real}(\text{card-slice } ((\lambda w. H w s) i) > \beta * \mu i)\} \geq (1-\beta)^2 * (\mu i)$   
**using** *pos-mu* **by** *force*  
**then have**  
 $(1 + (1-\beta)^2 * (\mu i)) * (1 - \text{measure-pmf.prob } p \{s. \text{real}(\text{card-slice } ((\lambda w. H w s) i) \leq \beta * \mu i)\}) \geq (1-\beta)^2 * (\mu i)$   
**using** *comp* **by** *auto*  
**then have**  
 $(1 + (1-\beta)^2 * (\mu i)) - (1 + (1-\beta)^2 * (\mu i)) * \text{measure-pmf.prob } p \{s. \text{real}(\text{card-slice } ((\lambda w. H w s) i) \leq \beta * \mu i)\} \geq (1-\beta)^2 * (\mu i)$   
**by** (*simp add: right-diff-distrib*)  
**then have**  
 $(1 + (1-\beta)^2 * (\mu i)) * \text{measure-pmf.prob } p \{s. \text{real}(\text{card-slice } ((\lambda w. H w s) i) \leq \beta * \mu i)\} \leq 1$   
**by** *simp*  
**thus** *?thesis* **by** (*smt* (*verit*, *best*) *mult-nonneg-nonneg nonzero-mult-div-cancel-left pos-mu real-divide-pos-left zero-le-power2*)  
**qed**

**lemma** *shift- $\mu$* :  
**assumes**  $k \leq i$   
**shows**  $\mu i * 2^k = \mu (i-k)$   
**unfolding**  *$\mu$ -def*  
**by** (*auto simp add: assms power-diff*)

**lemma** *analysis-2-1*:  
**assumes** *p: finite* (*set-pmf* *p*)  
**assumes** *ind: prob-space.k-universal* (*measure-pmf* *p*) 2 *H*  
 $\{\alpha::'a \text{ assg. dom } \alpha = S\}$   
 $\{\alpha::\text{nat assg. dom } \alpha = \{0..<\text{card } S - 1\}\}$   
**assumes**  *$\varepsilon$ -up*:  $\varepsilon \leq 1$   
**shows**  
 $\text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (T (mstar-3)) \leq 1 / 62.5$

**proof** –  
**have**  $1 + 1 / \varepsilon > 0$   
**by** (*simp add:  $\varepsilon$  pos-add-strict*)  
**then have** *pos-pivot*: *pivot* > 0  
**unfolding** *pivot-def*  
**by** *simp*

**have**  $mstar \geq 4 \vee mstar < 4$  **by** *auto*  
**moreover** {

```

assume  $mstar < 4$ 
then have  $measure-pmf.prob (map-pmf (\lambda s w. H w s) p) (T$ 
 $(mstar-3))$ 
   $\leq 1 / 62.5$ 
  by (auto simp add: T0-empty)
}
moreover {
  assume  $lo-mstar: mstar \geq 4$ 

  have  $extend-mu3: \mu (mstar-1) * 2^2 = \mu (mstar-3)$ 
  apply (subst shift- $\mu$ )
  subgoal using  $lo-mstar$  by linarith
  using numeral-3-eq-3
  using Suc-1 diff-Suc-eq-diff-pred numeral-nat(7) by presburger

  have  $*****: 1 + \varepsilon / (1 + \varepsilon) \leq 3 / 2$ 
  using  $\varepsilon$  assms(3) by auto
  have  $mu-mstar-3-gt-zero: \mu (mstar - 3) / 4 > 0$ 
  using  $\mu$ -gt-zero by simp

  from  $mstar-prop(1)$ 
  have  $thresh < \mu (mstar - 1) * 2^2 / 4 * (1 + \varepsilon / (1 + \varepsilon))$ 
  by auto
  also have  $... = \mu (mstar - 3) / 4 * (1 + \varepsilon / (1 + \varepsilon))$ 
  unfolding  $extend-mu3$  by auto
  also have  $... \leq \mu (mstar - 3) / 4 * (3 / 2)$ 
  apply (intro mult-left-mono)
  using  $*****$   $mu-mstar-3-gt-zero$  by auto
  also have  $... = 3 / 8 * \mu (mstar - 3)$ 
  by auto
  finally have  $thresh2mu: thresh < 3 / 8 * \mu (mstar - 3)$  .

  have  $1 + \varepsilon / (1 + \varepsilon) > 0$ 
  by (simp add: add-nonneg-pos  $\varepsilon$ )
  then have  $\mu (mstar-1) > pivot$ 
  using  $mstar-prop(1)$   $thresh$ 
  by (smt (verit) nonzero-mult-div-cancel-right real-divide-pos-left)
  then have  $lo-mu-mstar-3: \mu (mstar-3) > 4*pivot$ 
  using  $extend-mu3$ 
  by simp

  have  $mstar-3: mstar-3 \leq card S - 1$ 
  using  $lo-mstar$ 
  using diff-le-self dual-order.trans mstar-prop(3) by blast

  have  $*$ :  $(5 / 8)^2 = ((25 / 64)::real)$ 
  by (auto simp add: field-simps)
  have  $**$ :  $1 + 25/64 * 4*pivot \leq 1 + 25 / 64 * \mu (mstar - 3)$ 
  using  $lo-mu-mstar-3$ 

```

```

    by auto

  have  $1 + 1/\varepsilon \geq 2$ 
    by (simp add:  $\varepsilon$  assms(3))
  then have  $(1 + 1/\varepsilon)^2 \geq 2^2$ 
    by (smt (verit) power-mono)
  then have ***:  $1 + 25/64 * 4 * 4 * 9.84 \leq 1 + 25 / 64 * 4 * pivot$ 
    unfolding pivot-def
    by auto
  have ***:  $1 + 25/64 * 4 * 4 * 9.84 > (0::real)$ 
    by simp

  have measure-pmf.prob p
    {s. real (card-slice ( $\lambda w. H w s$ ) (mstar - 3)) ≤
      3 / 8 *  $\mu$  (mstar - 3)}
    ≤ 1 / (1 + (1 - 3 / 8)^2 *  $\mu$  (mstar - 3))
    apply(intro analysis-1-2[OF p ind mstar-3])
    by auto
  also have ... = 1 / (1 + (5 / 8)^2 *  $\mu$  (mstar-3))
    by simp
  also have ... = 1 / (1 + 25 / 64 *  $\mu$  (mstar-3))
    unfolding * by auto
  also have ... ≤ 1 / (1 + 25/64 * 4 * pivot)
    using **
  by (metis Groups.add-ac(2) add-sign-intros(3) divide-nonneg-nonneg
    frac-le mult-right-mono mult-zero-left of-nat-0-le-iff of-nat-numeral or-
    der-le-less pos-pivot zero-less-one)
  also have ... ≤ 1 / (1 + 25/64 * 4 * 4 * 9.84)
    using *** ****
    by (smt (verit) frac-le)
  also have ... ≤ 1 / 62.5
    by simp
  finally have *****: measure-pmf.prob p
    {s. real (card-slice ( $\lambda w. H w s$ ) (mstar - 3)) ≤ 3 / 8 *  $\mu$  (mstar
    - 3)} ≤ 1 / 62.5
    by auto

  have measure-pmf.prob (map-pmf ( $\lambda s w. H w s$ ) p) (T (mstar-3))
    = measure-pmf.prob p {s. real (card-slice ( $\lambda w. H w s$ ) (mstar
    - 3)) < thresh}
    unfolding T-def
    by auto
  also have ... ≤ measure-pmf.prob p
    {s. real (card-slice ( $\lambda w. H w s$ ) (mstar - 3)) ≤ 3 / 8 *  $\mu$  (mstar
    - 3)}
    using thresh2mu by (auto intro!: measure-pmf.finite-measure-mono)
  also have ... ≤ 1 / 62.5
    using *****
    by auto

```

**finally have**  $\text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (T (mstar-3)) \leq 1 / 62.5 .$   
**}**

**ultimately show** *?thesis* **by auto**  
**qed**

**lemma** *analysis-2-1'*:

**assumes**  $p$ : *finite* (*set-pmf*  $p$ )

**assumes** *ind*: *prob-space.k-universal* (*measure-pmf*  $p$ ) 2  $H$

$\{\alpha::'a \text{ assg. dom } \alpha = S\}$

$\{\alpha::\text{nat assg. dom } \alpha = \{0..<\text{card } S - 1\}\}$

**shows**

$\text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (T (mstar-3))$   
 $\leq 1 / 10.84$

**proof** –

**have**  $1 + 1 / \varepsilon > 0$

**by** (*simp add:  $\varepsilon$  pos-add-strict*)

**then have** *pos-pivot: pivot*  $> 0$

**unfolding** *pivot-def*

**by** *simp*

**have**  $mstar \geq 4 \vee mstar < 4$  **by auto**

**moreover** {

**assume**  $mstar < 4$

**then have**  $\text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (T (mstar-3))$

$\leq 1 / 10.84$

**by** (*auto simp add: T0-empty*)

**}**

**moreover** {

**assume** *lo-mstar: mstar*  $\geq 4$

**have** *extend-mu3*:  $\mu (mstar-1) * 2^2 = \mu (mstar-3)$

**apply** (*subst shift- $\mu$* )

**subgoal using** *lo-mstar semiring-norm(87)* **by** *linarith*

**using** *diff-Suc-eq-diff-pred eval-nat-numeral(3)* **by** *presburger*

**have**  $\varepsilon / (1 + \varepsilon) < 1$

**using**  $\varepsilon$  **by auto**

**then have** *\*\*\*\*\*: 1 +  $\varepsilon / (1 + \varepsilon) < 2$*

**using**  $\varepsilon$  **by auto**

**have** *mu-mstar-3-gt-zero:  $\mu (mstar - 3) / 4 > 0$*

**using**  *$\mu$ -gt-zero* **by** *simp*

**from** *mstar-prop(1)*

**have** *thresh*  $< \mu (mstar - 1) * (1 + \varepsilon / (1 + \varepsilon))$

**by auto**

**also have** ... =  $\mu (mstar - 1) * 2^2 / 4 * (1 + \varepsilon / (1 + \varepsilon))$   
**by** *auto*  
**also have** ... =  $\mu (mstar - 3) / 4 * (1 + \varepsilon / (1 + \varepsilon))$   
**unfolding** *extend-mu3* **by** *auto*  
**also have** ... <  $\mu (mstar - 3) / 4 * 2$   
**apply** (*intro mult-strict-left-mono*)  
**using** \*\*\*\*\* *mu-mstar-3-gt-zero* **by** *auto*  
**also have** ... =  $1 / 2 * \mu (mstar - 3)$   
**by** *auto*  
**finally have** *thresh2mu*: *thresh* <  $1 / 2 * \mu (mstar - 3)$  .

**have**  $1 + \varepsilon / (1 + \varepsilon) > 0$   
**by** (*simp add: add-nonneg-pos*  $\varepsilon$ )  
**then have**  $\mu (mstar - 1) * 4 > 4 * pivot$   
**using** *mstar-prop(1) thresh*  
**by** (*smt (verit) nonzero-mult-div-cancel-right real-divide-pos-left*)  
**then have** *lo-mu-mstar-3*:  $\mu (mstar - 3) > 4 * pivot$   
**using** *extend-mu3*  
**by** *simp*

**have** *mstar-3*:  $mstar - 3 \leq card\ S - 1$   
**using** *lo-mstar*  
**using** *diff-le-self dual-order.trans mstar-prop(3)* **by** *blast*

**have** \*:  $(1 / 2)^2 = ((1 / 4)::real)$   
**by** (*auto simp add: field-simps*)  
**have** \*\*:  $1 + 1/4 * 4 * pivot \leq 1 + 1/4 * \mu (mstar - 3)$   
**using** *lo-mu-mstar-3*  
**by** *auto*

**have**  $(1 + 1/\varepsilon)^2 > 1^2$   
**by** (*simp add:*  $\varepsilon$ *)*  
**then have** \*\*\*:  $1 + 1/4 * 4 * 9.84 \leq 1 + 1/4 * 4 * pivot$   
**unfolding** *pivot-def* **by** *auto*  
**have** \*\*\*\*:  $1 + 1/4 * 4 * 9.84 > (0::real)$   
**by** *simp*

**have** *measure-pmf.prob p*  
 $\{s. real (card\ slice (\lambda w. H\ w\ s) (mstar - 3)) \leq$   
 $1 / 2 * \mu (mstar - 3)\}$   
 $\leq 1 / (1 + (1 - 1 / 2)^2 * \mu (mstar - 3))$   
**apply** (*intro analysis-1-2[OF p ind mstar-3]*)  
**by** *auto*  
**also have** ... =  $1 / (1 + 1 / 4 * \mu (mstar - 3))$   
**using** \* **by** *force*  
**also have** ...  $\leq 1 / (1 + 1/4 * 4 * pivot)$   
**using** \*\*  
**by** (*simp add: add-nonneg-pos frac-le pos-pivot*)  
**also have** ...  $\leq 1 / (1 + 1/4 * 4 * 9.84)$



```

    using *** ****
    by (smt (verit) frac-le)
  also have ... ≤ 1 / 10.84
    by simp
  finally have *****: measure-pmf.prob p
    {s. real (card-slice (λw. H w s) (mstar - 3)) ≤ 1/2 * μ (mstar
- 3)} ≤ 1 / 10.84 .

  have measure-pmf.prob (map-pmf (λs w. H w s) p) (T (mstar-3))
    = measure-pmf.prob p {s. real (card-slice (λw. H w s) (mstar
- 3)) < thresh}
    unfolding T-def
    by auto
  also have ... ≤ measure-pmf.prob p
    {s. real (card-slice (λw. H w s) (mstar - 3)) ≤ 1 / 2 * μ (mstar
- 3)}
    using thresh2mu by (auto intro!: measure-pmf.finite-measure-mono)
  also have ... ≤ 1 / 10.84
    using *****
    by auto
  finally have measure-pmf.prob (map-pmf (λs w. H w s) p) (T
(mstar-3)) ≤ 1 / 10.84 .
}

ultimately show ?thesis by auto
qed

```

lemma analysis-2-2:

```

  assumes p: finite (set-pmf p)
  assumes ind: prob-space.k-universal (measure-pmf p) 2 H
    {α::'a assg. dom α = S}
    {α::nat assg. dom α = {0.. $\text{card } S - 1$ }}
  shows
    measure-pmf.prob (map-pmf (λs w. H w s) p) (L (mstar-2)) ≤ 1
/ 20.68
  proof -
    have epos: 1 + 1 / ε > 0
      by (simp add: ε pos-add-strict)
    then have pos-pivot: pivot > 0
      unfolding pivot-def
      by simp

    have mstar ≥ 3 ∨ mstar < 3 by auto
    moreover {
      assume mstar < 3
      then have measure-pmf.prob (map-pmf (λs w. H w s) p) (L
(mstar-2))
        ≤ 1 / 20.68
    }
  qed

```

```

    by (auto simp add: L0-empty)
  }
  moreover {
    assume lo-mstar: mstar ≥ 3

    have extend-mu2: μ (mstar-1) * 2^1 = μ (mstar-2)
      apply (subst shift-μ)
      subgoal using lo-mstar by linarith
      by (metis diff-diff-left one-add-one)

    have 1 + ε / (1 + ε) > 0
      by (simp add: add-nonneg-pos ε)
    then have μ (mstar-1) > pivot
      using mstar-prop(1) thresh
      by (smt (verit) nonzero-mult-div-cancel-right real-divide-pos-left)
    then have lo-mu-mstar-2: μ (mstar-2) > 2*pivot
      using extend-mu2
      by simp

    have mstar-2: mstar-2 ≤ card S - 1
      using lo-mstar
      using diff-le-self dual-order.trans mstar-prop(3) by blast

    have beta: 1/(1+ε) ≤ (1::real)
      using ε by auto
    have pos: (1 / (1 + 1/ε))^2 > 0
      using epos by auto
    then have *: 1 + (1 / (1 + 1/ε))^2 * 2*pivot ≤ 1 + (1 / (1 +
    1/ε))^2 * μ (mstar - 2)
      using lo-mu-mstar-2
      by auto

    have 1 - 1/(1+ε) = 1 / (1 + 1/ε)
      by (smt (verit, ccfv-threshold) ε conjugate-exponent-def conjugate-exponent-real(1))
    then have **: (1 - 1/(1+ε))^2 = (1 / (1 + 1/ε))^2
      by simp
    have ***: 1 + (1 / (1 + 1/ε))^2 * 2 * 9.84 * (1 + 1/ε)^2
      = 1 + (1 / (1 + 1/ε))^2 * (1 + 1/ε)^2 * 2 * 9.84
      by (simp add: mult.commute)
    have ****: (1 / (1 + 1/ε))^2 * (1 + 1/ε)^2 = 1
      using pos
      by (simp add: power-one-over)

    from analysis-1-2[OF p ind mstar-2 beta]
    have measure-pmf.prob p
      {s. real (card-slice (λw. H w s) (mstar - 2)) ≤ 1/(1+ε) * μ
      (mstar - 2)}
      ≤ 1 / (1 + (1 - 1/(1+ε))^2 * μ (mstar - 2))
  }

```

```

    by auto
    also have ... = 1 / (1 + (1 / (1 + 1/ε))2 * μ (mstar - 2))
    unfolding ** by auto
    also have ... ≤ 1 / (1 + (1 / (1 + 1/ε))2 * 2 * pivot)
    using *
    by (smt (verit) epos divide-pos-pos frac-le pivot-def pos-prod-le
zero-less-power)
    also have ... = 1 / (1 + (1 / (1 + 1/ε))2 * 2 * 9.84 * (1 +
1/ε)2)
    unfolding pivot-def by auto
    also have ... = 1 / 20.68
    unfolding *** **** by auto
    finally have *****: measure-pmf.prob p
    {s. real (card-slice (λw. H w s) (mstar - 2)) ≤ 1/(1+ε) * μ
(mstar - 2)} ≤ 1 / 20.68 .

    have measure-pmf.prob p
    {s. real (card-slice (λw. H w s) (mstar - 2)) < 1/(1+ε) * μ
(mstar - 2)}
    ≤ measure-pmf.prob p {s. real (card-slice (λw. H w s) (mstar
- 2)) ≤ 1/(1+ε) * μ (mstar - 2)}
    by (auto intro!: measure-pmf.finite-measure-mono)
    then have *****: measure-pmf.prob p
    {s. real (card-slice (λw. H w s) (mstar - 2)) < 1/(1+ε) * μ
(mstar - 2)}
    ≤ 1 / 20.68
    using ***** by auto

    have measure-pmf.prob (map-pmf (λs w. H w s) p) (L (mstar-2))
    = measure-pmf.prob p
    {s. real (card-slice (λw. H w s) (mstar - 2)) < 1/(1+ε) * μ
(mstar - 2)}
    unfolding L-def
    by auto
    also have ... ≤ 1 / 20.68
    using ***** by auto
    finally have measure-pmf.prob (map-pmf (λs w. H w s) p) (L
(mstar-2)) ≤ 1 / 20.68 .
}

ultimately show ?thesis by auto
qed

```

lemma analysis-2-3:

```

assumes p: finite (set-pmf p)
assumes ind: prob-space.k-universal (measure-pmf p) 2 H
  {α::'a assg. dom α = S}
  {α::nat assg. dom α = {0..<card S - 1}}

```

```

shows
  measure-pmf.prob
  (map-pmf ( $\lambda s w. H w s$ ) p) (L (mstar-1))  $\leq 1 / 10.84$ 
proof –
  have epos:  $1 + 1 / \varepsilon > 0$ 
    by (simp add:  $\varepsilon$  pos-add-strict)
  then have pos-pivot: pivot  $> 0$ 
    unfolding pivot-def
    by simp

  have mstar  $\geq 2 \vee mstar < 2$  by auto
  moreover {
    assume mstar  $< 2$ 
    then have measure-pmf.prob (map-pmf ( $\lambda s w. H w s$ ) p) (L
      (mstar-2))
       $\leq 1 / 10.84$ 
      by (auto simp add: L0-empty)
  }
  moreover {
    assume lo-mstar: mstar  $\geq 2$ 

    have  $1 + \varepsilon / (1 + \varepsilon) > 0$ 
      by (simp add: add-nonneg-pos  $\varepsilon$ )
    then have lo-mu-mstar-1:  $\mu$  (mstar-1)  $> pivot$ 
      using mstar-prop(1) using thresh
      by (smt (verit) nonzero-mult-div-cancel-right real-divide-pos-left)

    have mstar-1: mstar-1  $\leq card S - 1$ 
      using lo-mstar diff-le-self dual-order.trans mstar-prop(3) by blast

    have beta:  $1/(1+\varepsilon) \leq (1::real)$ 
      using  $\varepsilon$  by auto
    have  $(1 / (1 + 1/\varepsilon))^2 > 0$ 
      using epos by auto
    then have  $*$ :  $1 + (1 / (1 + 1/\varepsilon))^2 * pivot \leq 1 + (1 / (1 + 1/\varepsilon))^2 * \mu$  (mstar - 1)
      using lo-mu-mstar-1
      by auto

    have  $1 - 1/(1+\varepsilon) = 1 / (1 + 1/\varepsilon)$ 
      by (smt (verit, ccfv-threshold)  $\varepsilon$  conjugate-exponent-def conjugate-exponent-real(1))
    then have  $**$ :  $(1 - 1/(1+\varepsilon))^{\wedge 2} = (1 / (1 + 1/\varepsilon))^{\wedge 2}$ 
      by simp
    have  $***$ :  $1 + (1 / (1 + 1/\varepsilon))^2 * 9.84 * (1 + 1/\varepsilon)^{\wedge 2}$ 
       $= 1 + (1 / (1 + 1/\varepsilon))^2 * (1 + 1/\varepsilon)^{\wedge 2} * 9.84$ 
      by (simp add: mult.commute)
    have  $****$ :  $(1 / (1 + 1/\varepsilon))^2 * (1 + 1/\varepsilon)^{\wedge 2} = 1$ 
      by (metis (mono-tags, opaque-lifting)  $\langle 0 < 1 + 1 / \varepsilon \rangle$  div-by-1 di

```

*vide-divide-eq-right divide-self-if less-irrefl mult.commute power-mult-distrib power-one)*

```

from analysis-1-2[OF p ind mstar-1 beta]
have measure-pmf.prob p
  {s. real (card-slice (λw. H w s) (mstar - 1)) ≤ 1/(1+ε) * μ
(mstar - 1)}
  ≤ 1 / (1 + (1 - 1/(1+ε))2 * μ (mstar - 1))
  by auto
moreover have ... = 1 / (1 + (1 / (1 + 1/ε))2 * μ (mstar -
1))
  unfolding ** by auto
moreover have ... ≤ 1 / (1 + (1 / (1 + 1/ε))2 * pivot)
  using *
  by (smt (verit) <0 < (1 / (1 + 1 / ε))2> frac-le pos-pivot
zero-le-mult-iff)
moreover have ... = 1 / (1 + (1 / (1 + 1/ε))2 * 9.84 * (1 +
1/ε)2)
  unfolding pivot-def by auto
moreover have ... = 1 / (1 + (1 / (1 + 1/ε))2 * (1 + 1/ε)2
* 9.84)
  unfolding *** by auto
moreover have ... = 1 / (1 + 9.84)
  unfolding **** by auto
moreover have ... = 1 / 10.84
  by auto
ultimately have *****: measure-pmf.prob p
  {s. real (card-slice (λw. H w s) (mstar - 1)) ≤ 1/(1+ε) * μ
(mstar - 1)} ≤ 1 / 10.84
  by linarith
have measure-pmf.prob p
  {s. real (card-slice (λw. H w s) (mstar - 1)) < 1/(1+ε) * μ
(mstar - 1)}
  ≤ measure-pmf.prob p
  {s. real (card-slice (λw. H w s) (mstar - 1)) ≤ 1/(1+ε) * μ
(mstar - 1)}
  by (auto intro!: measure-pmf.finite-measure-mono)
then have *****: measure-pmf.prob p
  {s. real (card-slice (λw. H w s) (mstar - 1)) < 1/(1+ε) * μ
(mstar - 1)}
  ≤ 1 / 10.84
  using ***** by auto

have measure-pmf.prob (map-pmf (λs w. H w s) p) (L (mstar-1))
  = measure-pmf.prob p
  {s. real (card-slice (λw. H w s) (mstar - 1)) < 1/(1+ε) * μ
(mstar - 1)}
  unfolding L-def
  by auto

```

```

    moreover have ... ≤ 1 / 10.84
      using ***** by auto
    ultimately have measure-pmf.prob (map-pmf (λs w. H w s) p)
      (L (mstar-1)) ≤ 1 / 10.84
      by auto
  }

  ultimately show ?thesis by auto
qed

```

```

lemma analysis-2-4:
  assumes p: finite (set-pmf p)
  assumes ind: prob-space.k-universal (measure-pmf p) 2 H
    {α::'a assg. dom α = S}
    {α::nat assg. dom α = {0..card S - 1}}
  shows
    measure-pmf.prob (map-pmf (λs w. H w s) p) (L mstar ∪ U mstar)
      ≤ 1 / 4.92
  proof -
    have 1 + 1 / ε > 0
      by (simp add: ε pos-add-strict)
    then have pos-pivot: pivot > 0
      unfolding pivot-def
      by simp

    have mstar ≥ 1 ∨ mstar < 1 by auto
    moreover {
      assume mstar < 1
      have LU0-empty: L mstar ∪ U mstar = {}
        using L0-empty U0-empty
        using ⟨mstar < 1⟩ less-one by auto
      then have measure-pmf.prob (map-pmf (λs w. H w s) p) (L mstar
        ∪ U mstar)
        ≤ 1 / 4.92
        by auto
    }
    moreover {
      assume lo-mstar: mstar ≥ 1
      have extend-mu: μ mstar * 2^1 = μ (mstar-1)
        using lo-mstar
        apply (subst shift-μ)
        by auto

      have 1 + ε / (1 + ε) > 0
        by (simp add: add-nonneg-pos ε)
      then have μ (mstar-1) > pivot
        using mstar-prop(1) thresh
        by (smt (verit) nonzero-mult-div-cancel-right real-divide-pos-left)
    }
  qed

```

```

then have lo-mu-star:  $\mu \text{ mstar} > \text{pivot} / 2$ 
  using extend-mu
  by auto

have mstar:  $\text{mstar} \leq \text{card } S - 1$ 
  using lo-mstar
  using diff-le-self dual-order.trans mstar-prop(3) by blast

have  $\varepsilon * (1 + 1/\varepsilon) = 1 + \varepsilon$ 
  by (smt (verit)  $\varepsilon$  add-divide-eq-if-simps(1) divide-cancel-right
  divide-self-if nonzero-mult-div-cancel-left)
  then have *:  $9.84 * \varepsilon^2 * (1 + 1/\varepsilon)^2 / 2 = 9.84 * (1 + \varepsilon)^2 / 2$ 
  by (metis (mono-tags, opaque-lifting) more-arith-simps(11) power-mult-distrib)

have  $\varepsilon^2 * \mu \text{ mstar} \geq \varepsilon^2 * \text{pivot} / 2$ 
  using lo-mu-star
  by (metis less-eq-real-def ordered-comm-semiring-class.comm-mult-left-mono
  times-divide-eq-right zero-le-power2)
  then have **:  $\varepsilon^2 * \mu \text{ mstar} \geq 9.84 * (1 + \varepsilon)^2 / 2$ 
  unfolding pivot-def using * by auto

from analysis-1-1[OF p ind mstar]
have measure-pmf.prob p
  {s. | card-slice (( $\lambda w. H w s$ )) mstar -  $\mu \text{ mstar}$  |  $\geq \varepsilon / (1 + \varepsilon) * \mu \text{ mstar}$  }
   $\leq (1 + \varepsilon)^2 / (\varepsilon^2 * \mu \text{ mstar})$ 
  by auto
  also have ...  $\leq (1 + \varepsilon)^2 / (9.84 * (1 + \varepsilon)^2 / 2)$ 
  using **
  by (smt (verit) divide-cancel-left divide-le-0-iff frac-le pos-prod-le
  power2-less-0 zero-eq-power2)
  also have ...  $= (1 + \varepsilon)^2 / (4.92 * (1 + \varepsilon)^2)$ 
  by simp
  also have ...  $= 1 / 4.92$ 
  using  $\varepsilon$  by simp
  finally have *****: measure-pmf.prob p
  {s. | card-slice (( $\lambda w. H w s$ )) mstar -  $\mu \text{ mstar}$  |  $\geq \varepsilon / (1 + \varepsilon) * \mu \text{ mstar}$  }
   $\leq 1 / 4.92$  .

have  $\mu \text{ mstar} / (1 + \varepsilon) - \mu \text{ mstar} = \mu \text{ mstar} * (1 / (1 + \varepsilon) - 1)$ 
  by (simp add: right-diff-distrib')
  also have ...  $= \mu \text{ mstar} * (-\varepsilon / (1 + \varepsilon))$ 
  by (smt (verit)  $\varepsilon$  add-divide-distrib div-self minus-divide-left)
  finally have ***:  $\mu \text{ mstar} / (1 + \varepsilon) - \mu \text{ mstar} = \mu \text{ mstar} * (-\varepsilon / (1 + \varepsilon))$  .

have {h. real (card-slice h mstar)  $\leq \mu \text{ mstar} / (1 + \varepsilon)$ }
  = {h. real (card-slice h mstar) -  $\mu \text{ mstar} \leq \mu \text{ mstar} / (1 + \varepsilon)$ }

```

$-\mu \text{ mstar}$   
**by auto**  
**also have** ... =  $\{h. \text{ real } (\text{card-slice } h \text{ mstar}) - \mu \text{ mstar} \leq \mu \text{ mstar}$   
 $* (-\varepsilon / (1+\varepsilon))\}$   
**using \*\*\* by auto**  
**finally have \*\*\*\*:**  
 $\{h. \text{ real } (\text{card-slice } h \text{ mstar}) \leq \mu \text{ mstar} / (1 + \varepsilon)\} =$   
 $\{h. \text{ real } (\text{card-slice } h \text{ mstar}) - \mu \text{ mstar} \leq \mu \text{ mstar} * (-\varepsilon /$   
 $(1+\varepsilon))\}$  .

**have**  $L \text{ mstar}$   
 $= \{h. \text{ real } (\text{card-slice } h \text{ mstar}) < \mu \text{ mstar} / (1 + \varepsilon)\}$   
**unfolding**  $L\text{-def}$  **by auto**  
**also have** ...  $\subseteq \{h. \text{ real } (\text{card-slice } h \text{ mstar}) \leq \mu \text{ mstar} / (1 + \varepsilon)\}$   
**by auto**  
**also have** ... =  $\{h. \text{ real } (\text{card-slice } h \text{ mstar}) - \mu \text{ mstar} \leq \mu \text{ mstar}$   
 $* (-\varepsilon / (1+\varepsilon))\}$   
**unfolding \*\*\*\* by auto**  
**finally have**  $\text{extend-L: } L \text{ mstar} \subseteq \{h. \text{ real } (\text{card-slice } h \text{ mstar}) -$   
 $\mu \text{ mstar} \leq \mu \text{ mstar} * (-\varepsilon / (1+\varepsilon))\}$  .

**have \*\*\*\*\*:**  $\mu \text{ mstar} * (1 + \varepsilon / (1+\varepsilon)) - \mu \text{ mstar} = \mu \text{ mstar} *$   
 $(\varepsilon / (1+\varepsilon))$   
**by** (*metis (no-types, opaque-lifting) add commute diff-add-cancel*  
*group-cancel.sub1 mult.right-neutral right-diff-distrib'*)

**have \*\*\*\*\*:**  $U \text{ mstar} = \{h. \text{ real } (\text{card-slice } h \text{ mstar}) \geq \mu \text{ mstar}$   
 $* (1 + \varepsilon / (1+\varepsilon))\}$   
**unfolding**  $U\text{-def}$  **by auto**  
**have**  $\{h. \text{ real } (\text{card-slice } h \text{ mstar}) \geq \mu \text{ mstar} * (1 + \varepsilon / (1+\varepsilon))\}$   
 $= \{h. \text{ real } (\text{card-slice } h \text{ mstar}) - \mu \text{ mstar} \geq \mu \text{ mstar} * (1 +$   
 $\varepsilon / (1+\varepsilon)) - \mu \text{ mstar}\}$   
**by auto**  
**also have** ... =  $\{h. \text{ real } (\text{card-slice } h \text{ mstar}) - \mu \text{ mstar} \geq \mu \text{ mstar}$   
 $* (\varepsilon / (1+\varepsilon))\}$   
**unfolding \*\*\*\*\* by auto**  
**finally have**  $\text{extend-U: } U \text{ mstar} = \{h. \text{ real } (\text{card-slice } h \text{ mstar}) -$   
 $\mu \text{ mstar} \geq \mu \text{ mstar} * (\varepsilon / (1+\varepsilon))\}$   
**using \*\*\*\*\* by auto**

**have**  $L \text{ mstar} \cup U \text{ mstar} \subseteq$   
 $\{h. \text{ real } (\text{card-slice } h \text{ mstar}) - \mu \text{ mstar} \leq \mu \text{ mstar} *$   
 $(-\varepsilon / (1+\varepsilon))\}$   
 $\cup \{h. \text{ real } (\text{card-slice } h \text{ mstar}) - \mu \text{ mstar} \geq \mu \text{ mstar} *$   
 $(\varepsilon / (1+\varepsilon))\}$   
**unfolding**  $\text{extend-U}$   
**using**  $\text{extend-L}$   
**by auto**  
**also have** ... =  $\{h. | \text{ real } (\text{card-slice } h \text{ mstar}) - \mu \text{ mstar} | \geq \mu$



$mstar * (\varepsilon / (1+\varepsilon))\}$   
**by** *auto*  
**finally have** *extend-LU*:  $L\ mstar \cup U\ mstar \subseteq \{h. \mid \text{real}(\text{card-slice } h\ mstar) - \mu\ mstar \mid \geq \mu\ mstar * (\varepsilon / (1+\varepsilon))\}$  .  
  
**have** *measure-pmf.prob* (*map-pmf* ( $\lambda s\ w.\ H\ w\ s$ )  $p$ ) ( $L\ mstar \cup U\ mstar$ )  
 $\leq \text{measure-pmf.prob}(\text{map-pmf}(\lambda s\ w.\ H\ w\ s)\ p)\ \{h. \mid \text{real}(\text{card-slice } h\ mstar) - \mu\ mstar \mid \geq \mu\ mstar * (\varepsilon / (1+\varepsilon))\}$   
**using** *extend-LU*  
**by** (*auto intro!*: *measure-pmf.finite-measure-mono*)  
**also have** ... = *measure-pmf.prob*  $p$   
 $\{s. \mid \text{real}(\text{card-slice } (\lambda w.\ H\ w\ s)\ mstar) - \mu\ mstar \mid \geq \varepsilon / (1 + \varepsilon) * \mu\ mstar\}$   
**by** (*simp add: mult.commute*)  
**also have** ...  $\leq 1 / 4.92$   
**using** \*\*\*\*\* **by** *auto*  
**finally have** *measure-pmf.prob* (*map-pmf* ( $\lambda s\ w.\ H\ w\ s$ )  $p$ ) ( $L\ mstar \cup U\ mstar$ )  $\leq 1 / 4.92$  .  
**}**  
  
**ultimately show** *?thesis* **by** *auto*  
**qed**

**lemma** *analysis-3*:

**assumes**  $p$ : *finite* (*set-pmf*  $p$ )  
**assumes** *ind*: *prob-space.k-universal* (*measure-pmf*  $p$ )  $2\ H$   
 $\{\alpha::'a\ \text{assg.}\ \text{dom}\ \alpha = S\}$   
 $\{\alpha::\text{nat}\ \text{assg.}\ \text{dom}\ \alpha = \{0..\text{card } S - 1\}\}$   
**assumes**  $\varepsilon$ -*up*:  $\varepsilon \leq 1$   
**shows**  
 $\text{measure-pmf.prob}(\text{map-pmf}(\lambda s\ w.\ H\ w\ s)\ p)$   
 $\text{approxcore-fail} \leq 0.36$

**proof** –

**have** *measure-pmf.prob* (*map-pmf* ( $\lambda s\ w.\ H\ w\ s$ )  $p$ ) *approxcore-fail*  
 $\leq \text{measure-pmf.prob}(\text{map-pmf}(\lambda s\ w.\ H\ w\ s)\ p)$   
 $(T\ (mstar-3) \cup$   
 $L\ (mstar-2) \cup$   
 $L\ (mstar-1) \cup$   
 $(L\ mstar \cup U\ mstar))$   
**using** *failure-bound*  
**by** (*auto intro!*: *measure-pmf.finite-measure-mono*)

**moreover have** ...  $\leq$

$\text{measure-pmf.prob}(\text{map-pmf}(\lambda s\ w.\ H\ w\ s)\ p)\ (T\ (mstar-3) \cup$   
 $L\ (mstar-2) \cup L\ (mstar-1))$   
 $+ \text{measure-pmf.prob}(\text{map-pmf}(\lambda s\ w.\ H\ w\ s)\ p)\ (L\ mstar \cup U\ mstar)$

**by** (*auto intro!*: *measure-pmf.finite-measure-subadditive*)  
**moreover have** ...  $\leq$   
 $\text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (T (mstar-3) \cup$   
 $L (mstar-2))$   
 $+ \text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (L (mstar-1))$   
 $+ \text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (L mstar \cup U$   
 $mstar)$   
**by** (*auto intro!*: *measure-pmf.finite-measure-subadditive*)  
**moreover have** ...  $\leq$   
 $\text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (T (mstar-3))$   
 $+ \text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (L (mstar-2))$   
 $+ \text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (L (mstar-1))$   
 $+ \text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (L mstar \cup U$   
 $mstar)$   
**by** (*auto intro!*: *measure-pmf.finite-measure-subadditive*)  
**moreover have** ...  $\leq 1/62.5 + 1/20.68 + 1/10.84 + 1/4.92$   
**using** *analysis-2-1*[*OF p ind  $\varepsilon$ -up*]  
**using** *analysis-2-2*[*OF p ind*]  
**using** *analysis-2-3*[*OF p ind*]  
**using** *analysis-2-4*[*OF p ind*]  
**by** *auto*  
**ultimately show** *?thesis by force*  
**qed**

**lemma** *analysis-3'*:

**assumes** *p*: *finite* (*set-pmf p*)  
**assumes** *ind*: *prob-space.k-universal* (*measure-pmf p*) 2 *H*  
 $\{\alpha::'a \text{ assg. } \text{dom } \alpha = S\}$   
 $\{\alpha::\text{nat assg. } \text{dom } \alpha = \{0..<\text{card } S - 1\}\}$

**shows**

$\text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p)$   
 $\text{approxcore-fail} \leq 0.44$

**proof** –

**have**  $\text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) \text{approxcore-fail}$   
 $\leq \text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p)$   
 $(T (mstar-3) \cup$   
 $L (mstar-2) \cup$   
 $L (mstar-1) \cup$   
 $(L mstar \cup U mstar))$   
**using** *failure-bound*  
**by** (*auto intro!*: *measure-pmf.finite-measure-mono*)

**moreover have** ...  $\leq$

$\text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (T (mstar-3) \cup$   
 $L (mstar-2) \cup L (mstar-1))$   
 $+ \text{measure-pmf.prob } (\text{map-pmf } (\lambda s w. H w s) p) (L mstar \cup U$   
 $mstar)$   
**by** (*auto intro!*: *measure-pmf.finite-measure-subadditive*)

```

moreover have ...  $\leq$ 
  measure-pmf.prob (map-pmf ( $\lambda s w. H w s$ )  $p$ ) (T (mstar-3))  $\cup$ 
L (mstar-2))
  + measure-pmf.prob (map-pmf ( $\lambda s w. H w s$ )  $p$ ) (L (mstar-1))
  + measure-pmf.prob (map-pmf ( $\lambda s w. H w s$ )  $p$ ) (L mstar  $\cup$  U
mstar)
  by (auto intro!: measure-pmf.finite-measure-subadditive)
moreover have ...  $\leq$ 
  measure-pmf.prob (map-pmf ( $\lambda s w. H w s$ )  $p$ ) (T (mstar-3))
  + measure-pmf.prob (map-pmf ( $\lambda s w. H w s$ )  $p$ ) (L (mstar-2))
  + measure-pmf.prob (map-pmf ( $\lambda s w. H w s$ )  $p$ ) (L (mstar-1))
  + measure-pmf.prob (map-pmf ( $\lambda s w. H w s$ )  $p$ ) (L mstar  $\cup$  U
mstar)
  by (auto intro!: measure-pmf.finite-measure-subadditive)
moreover have ...  $\leq 1/10.84 + 1/20.68 + 1/10.84 + 1/4.92$ 
  using analysis-2-1 [OF p ind]
  using analysis-2-2 [OF p ind]
  using analysis-2-3 [OF p ind]
  using analysis-2-4 [OF p ind]
  by auto
ultimately show ?thesis by auto
qed
end

end

end

```

## 6 ApproxMC definition and analysis

This section puts together preceding results to formalize the PAC guarantee of ApproxMC.

```

theory ApproxMCAnalysis imports

```

```

  ApproxMCCoreAnalysis
  RandomXORHashFamily
  Median-Method.Median

```

```

begin

```

```

lemma replicate-pmf-Pi-pmf:

```

```

  assumes distinct ls
  shows replicate-pmf (length ls) P =
    map-pmf ( $\lambda f. \text{map } f \text{ } ls$ )
    (Pi-pmf (set ls) def ( $\lambda-. P$ ))

```

```

  using assms

```

```

proof (induction ls)

```

```

  case Nil

```

```

  then show ?case

```

```

  by auto

```

```

next
  case (Cons x xs)
  then show ?case
    by (auto intro!: bind-pmf-cong simp add: Pi-pmf-insert' map-bind-pmf
bind-map-pmf)
qed

```

```

lemma replicate-pmf-Pi-pmf':
  assumes finite V
  shows replicate-pmf (card V) P =
    map-pmf (λf. map f (sorted-list-of-set V))
    (Pi-pmf V def (λ-. P))
proof –
  have *: card V = length (sorted-list-of-set V)
    using assms by auto
  show ?thesis
    unfolding *
    apply (subst replicate-pmf-Pi-pmf)
    using assms by auto
qed

```

```

definition map-of-default::('a × 'b) list ⇒ 'b ⇒ 'a ⇒ 'b
where map-of-default ls def =
  (let m = map-of ls in
    (λx. case m x of None ⇒ def | Some v ⇒ v))

```

```

lemma Pi-pmf-replicate-pmf:
  assumes finite V
  shows
    (Pi-pmf V def (λ-. p)) =
    map-pmf (λbs.
      map-of-default (zip (sorted-list-of-set V) bs) def)
    (replicate-pmf (card V) p)
proof –
  show ?thesis
    apply (subst replicate-pmf-Pi-pmf'[OF assms, where def=def])
    unfolding map-pmf-comp
    apply (intro map-pmf-idI[symmetric])
    unfolding map-of-default-def Let-def fun-eq-iff map-of-zip-map
    by (smt (verit, del-insts) assms option.case(1) option.case(2) pmf-Pi
pmf-eq-0-set-pmf sorted-list-of-set.set-sorted-key-list-of-set)
qed

```

```

lemma proj-inter-neutral:
  assumes  $\bigwedge w. w \in B \longleftrightarrow \text{restr } S \ w \in C$ 
  shows proj S (A ∩ B) = proj S A ∩ C
  unfolding ApproxMCCore.proj-def
  using assms by auto

```

An abstract spec of ApproxMC for any Boolean theory. This lo-

cale must be instantiated with a theory implementing the two the functions below (and satisfying the assumption linking them).

```

locale ApproxMC =
  fixes sols :: 'fml  $\Rightarrow$  ('a  $\Rightarrow$  bool) set
  fixes enc-xor :: 'a set  $\times$  bool  $\Rightarrow$  'fml  $\Rightarrow$  'fml
  assumes sols-enc-xor:
     $\bigwedge F \text{ xor. finite (fst xor)} \implies$ 
      sols (enc-xor xor F) =
        sols F  $\cap$  { $\omega$ . satisfies-xor xor {x.  $\omega$  x}}
begin

definition compute-thresh :: real  $\Rightarrow$  nat
  where compute-thresh  $\varepsilon$  =
    nat  $\lceil 1 + 9.84 * (1 + \varepsilon / (1 + \varepsilon)) * (1 + 1 / \varepsilon)^2 \rceil$ 

definition fix-t :: real  $\Rightarrow$  nat
  where fix-t  $\delta$  =
    nat  $\lceil \ln (1 / \delta) / (2 * (0.5 - 0.36)^2) \rceil$ 

definition raw-median-bound :: real  $\Rightarrow$  nat  $\Rightarrow$  real
  where raw-median-bound  $\alpha$  t =
    ( $\sum i = 0..t \text{ div } 2$ .
      (t choose i) *  $(1 / 2 + \alpha)^i * (1 / 2 - \alpha)^{t-i}$ )

definition compute-t :: real  $\Rightarrow$  nat  $\Rightarrow$  nat
  where compute-t  $\delta$  n =
    (if raw-median-bound 0.14 n <  $\delta$  then n
     else fix-t  $\delta$ )

definition size-xor ::
  'fml  $\Rightarrow$  'a set  $\Rightarrow$ 
  (nat  $\Rightarrow$  ('a set  $\times$  bool) option)  $\Rightarrow$ 
  nat  $\Rightarrow$  nat
  where size-xor F S xorsf i = (
    let xors = map (the  $\circ$  xorsf) [0..i] in
    let Fenc = fold enc-xor xors F in
      card (proj S (sols Fenc))
  )

definition check-xor ::
  'fml  $\Rightarrow$  'a set  $\Rightarrow$ 
  nat  $\Rightarrow$ 
  (nat  $\Rightarrow$  ('a set  $\times$  bool) option)  $\Rightarrow$ 
  nat  $\Rightarrow$  bool
  where check-xor F S thresh xorsf i =
    (size-xor F S xorsf i < thresh)

definition approxcore-xors ::

```

```

'fml ⇒ 'a set ⇒
nat ⇒
(nat ⇒ ('a set × bool) option) ⇒
nat
where
  approxcore-xors F S thresh xorsf =
  (case List.find
    (check-xor F S thresh xorsf) [1..<card S] of
    None ⇒ 2 ^ card S
    | Some m ⇒
      (2 ^ m * size-xor F S xorsf m))

definition approxmccore :: 'fml ⇒ 'a set ⇒ nat ⇒ nat pmf
where approxmccore F S thresh =
  map-pmf (approxcore-xors F S thresh) (random-xors S (card S - 1))

definition approxmc :: 'fml ⇒ 'a set ⇒ real ⇒ real ⇒ nat ⇒ nat
pmf
where approxmc F S ε δ n = (
  let thresh = compute-thresh ε in
  if card (proj S (sols F)) < thresh then
    return-pmf (card (proj S (sols F)))
  else
    let t = compute-t δ n in
    map-pmf (median t)
      (prod-pmf {0..<t::nat} (λi. approxmccore F S thresh))
)

lemma median-commute:
assumes t ≥ 1
shows (real ∘ median t) = (λw::nat ⇒ nat. median t (real ∘ w))
unfolding median-def map-map[symmetric]
apply (subst Median.map-sort[where f = λx. real x])
subgoal by (clarsimp simp add:mono-def)
apply (subst nth-map)
using assms by auto

lemma median-default:
shows median t y = median t (λx. if x < t then y x else def)
unfolding median-def
by (auto intro!: arg-cong[where f = λls. sort ls ! (t div 2)])

definition default-α::'a set ⇒ nat assg
where default-α S i = (if i < card S - 1 then Some True else None)

lemma dom-default-α:
  dom (default-α S) = {0..<card S - 1}
by (auto simp add:default-α-def split: if-splits)

```

```

lemma compute-thresh-bound-4:
  assumes  $\varepsilon > 0$ 
  shows  $4 < \text{compute-thresh } \varepsilon$ 
proof -
  have 1:  $(1 + \varepsilon / (1 + \varepsilon)) > 1$ 
    using assms
    by simp
  have 2:  $(1 + 1 / \varepsilon)^2 > 1$ 
    using assms by simp

  define a where  $a = (1 + \varepsilon / (1 + \varepsilon)) * (1 + 1 / \varepsilon)^2$ 
  have  $a > 1$  unfolding a-def
    using 1 2
    using less-1-mult by blast
  then have  $(984 / 10^2) * a \geq 4$ 
    by auto

  thus ?thesis
    unfolding compute-thresh-def
    by (smt (verit) a-def arith-special(2) arithmetic-simps(1) more-arith-simps(11)
  nat-less-real-le numeral-Bit0 of-nat-numeral real-nat-ceiling-ge)
qed

```

```

lemma satisfies-xor-with-domain:
  assumes  $\text{fst } x \subseteq S$ 
  shows  $\text{satisfies-xor } x \{x. w x\} \longleftrightarrow$ 
     $\text{satisfies-xor } x (\{x. w x\} \cap S)$ 
  using assms
  apply (cases x)
  by (simp add: Int-assoc inf.absorb-iff2)

```

```

lemma approxcore-xors-eq:
  assumes thresh:
     $\text{thresh} = \text{compute-thresh } \varepsilon$ 
     $\text{thresh} \leq \text{card } (\text{proj } S (\text{sols } F))$ 
  assumes  $\varepsilon: \varepsilon > (0::\text{real}) \ \varepsilon \leq 1$ 
  assumes S: finite S
  shows  $\text{measure-pmf.prob } (\text{random-xors } S (\text{card } S - 1))$ 
     $\{ \text{xors. real } (\text{approxcore-xors } F S \text{ thresh } \text{xors}) \in$ 
       $\{ \text{real } (\text{card } (\text{proj } S (\text{sols } F))) / (1 + \varepsilon)..$ 
       $(1 + \varepsilon) * \text{real } (\text{card } (\text{proj } S (\text{sols } F))) \} \} \geq 0.64$ 
proof -

```

```

  have  $\text{ApproxMCCore } (\text{sols } F) S \varepsilon (\text{default-}\alpha S) \text{ thresh}$ 
    apply unfold-locales
    subgoal using dom-default-}\alpha by simp
    subgoal using  $\varepsilon$  by simp
    subgoal using thresh assms(3) compute-thresh-bound-4 by blast

```

```

using thresh  $S$  by auto
then interpret amc: ApproxMCCore sols  $F$  - - (default- $\alpha$   $S$ ) .

have
   $m < \text{card } S \implies$ 
   $\{0..<m\} \subseteq \text{dom } xors \implies$ 
   $(\bigwedge i. i < m \implies \text{fst } (\text{the } (xors \ i)) \subseteq S) \implies$ 
  (proj  $S$ 
    (sols (fold enc-xor (map (the  $\circ$  xors)  $[0..<m]$ )  $F$ ))) =
  proj  $S$  (sols  $F$ )  $\cap$ 
    { $w$ . hslice  $m$  ( $\lambda\omega$ . xor-hash  $\omega$  xors)  $w$  =
      aslice  $m$  (default- $\alpha$   $S$ )} for  $m$  xors
proof (induction  $m$ )
  case 0
  then show ?case
    unfolding hslice-def aslice-def
    by auto
next
  case (Suc  $m$ )
  have  $m: m \in \text{dom } xors$ 
    by (meson Set.basic-monos(7) Suc(3) atLeastLessThan-iff le0
lessI)
  have  $sp: \text{fst } (\text{the } (xors \ m)) \subseteq S$ 
    by (simp add: Suc(4))

  then obtain xor where  $x: xors \ m = \text{Some } xor$ 
    using  $m$  by blast

  have eq: { $w$ . xor-hash  $w$  xors  $m = \text{Some } \text{True}$ } =
    { $\omega$ . satisfies-xor xor { $x$ .  $\omega$   $x = \text{Some } \text{True}$ }}
    unfolding xor-hash-def
    by (clarsimp simp add:  $x$ )

  have neut:  $\bigwedge w$ .
     $w \in \{\omega$ . satisfies-xor xor { $x$ .  $\omega$   $x\}\longleftrightarrow$ 
    restr  $S$   $w \in \{\omega$ . satisfies-xor xor { $x$ .  $\omega$   $x = \text{Some } \text{True}\}$ }
    using  $sp$  unfolding restr-def
    by (smt (verit, ccfv-SIG) Collect-cong Int-def mem-Collect-eq
option.sel satisfies-xor-with-domain  $x$ )

  have lhs:
    proj  $S$ 
    (sols (fold enc-xor (map (the  $\circ$  xors)  $[0..<\text{Suc } m]$ )  $F$ )) =
    proj  $S$  (sols (fold enc-xor (map (the  $\circ$  xors)  $[0..<m]$ )  $F$ ))  $\cap$ 
    { $w$ . xor-hash  $w$  xors  $m = \text{Some } \text{True}$ }
    apply clarsimp
    apply (subst sols-enc-xor)
    subgoal using assms(5) rev-finite-subset  $sp$  by blast
    apply (subst proj-inter-neutral)

```



```

using eq neut x by auto

have rhs1: hslice (Suc m) (λω. xor-hash ω xors) w = aslice (Suc
m) (default-α S) ⇒
  hslice m (λω. xor-hash ω xors) w = aslice m (default-α S) for w
unfolding hslice-def aslice-def fun-eq-iff
by (auto simp add:lessThan-Suc restrict-map-def split: if-splits)

have rhs2:hslice (Suc m) (λω. xor-hash ω xors) w = aslice (Suc
m) (default-α S) ⇒
  xor-hash w xors m = Some True for w
unfolding hslice-def aslice-def fun-eq-iff
apply (clarsimp simp add:lessThan-Suc restrict-map-def )
by (metis default-α-def domIff m xor-hash-eq-dom)

have rhs3: hslice m (λω. xor-hash ω xors) w = aslice m (default-α
S) ⇒
  xor-hash w xors m = Some True ⇒
  hslice (Suc m) (λω. xor-hash ω xors) w = aslice (Suc m) (default-α
S) for w
unfolding hslice-def aslice-def fun-eq-iff
apply (clarsimp simp add:lessThan-Suc restrict-map-def )
by (metis One-nat-def Suc.premis(1) Suc-lessD Suc-less-eq Suc-pred
default-α-def gr-zeroI zero-less-diff)

have rhs: {w. hslice (Suc m) (λω. xor-hash ω xors) w = aslice (Suc
m) (default-α S)} =
  {w. hslice m (λω. xor-hash ω xors) w = aslice m (default-α S)}
∩
  {w. xor-hash w xors m = Some True}
by (auto simp add: rhs1 rhs2 rhs3)

have ih: proj S (sols (fold enc-xor (map (the ∘ xors) [0..apply (intro Suc(1))
using Suc(2) Suc-lessD apply blast
using Suc(3) atLeast0-lessThan-Suc apply blast
using Suc(4) less-SucI by blast

show ?case
unfolding lhs rhs
by (simp add: Int-ac(1) ih)
qed
then have *:
  m < card S ⇒
  {0..

```

*size-xor*  $F S$   $xors$   $m =$   
*amc.card-slice*  $(\lambda\omega. xor\text{-hash } \omega \ xors)$   $m$  **for**  $m$   $xors$   
**unfolding** *size-xor-def amc.card-slice-def*  
**by** *auto*

**have** \*\*:
   
 $\{0..<card\ S - 1\} \subseteq dom\ xors \implies$   
 $(\bigwedge i. i < card\ S - 1 \implies fst\ (the\ (xors\ i)) \subseteq S) \implies$   
 $find\ (check\text{-xor}\ F\ S\ thresh\ xors)\ [1..<card\ S] =$   
 $find\ (\lambda i. (\lambda\omega. xor\text{-hash } \omega \ xors) \in amc.T\ i)\ [1..<card\ S]$  **for**  $xors$   
**apply** *(intro find-cong)*  
**unfolding** *check-xor-def amc.T-def*  
**subgoal** **by** *simp*  
**apply** *(subst \*)*  
**subgoal** **by** *clarsimp*  
**subgoal** **by** *(clarsimp simp add: domIff subset-iff)*  
**by** *auto*

**have** *rw*:
   
 $\{0..<card\ S - 1\} \subseteq dom\ xors \implies$   
 $(\bigwedge i. i < card\ S - 1 \implies fst\ (the\ (xors\ i)) \subseteq S) \implies$   
 $approxcore\text{-xors}\ F\ S\ thresh\ xors =$   
 $(\lambda(a,b). a*b)\ (amc.approxcore\ (\lambda\omega. xor\text{-hash } \omega \ xors))$  **for**  $xors$   
**apply** *(subst amc.approxcore-def)*  
**unfolding** *approxcore-xors-def*  
**apply** *(subst \*\*)*  
**subgoal** **by** *clarsimp*  
**subgoal** **by** *clarsimp*  
**apply** *(clarsimp split: option.split)*  
**apply** *(subst \*)*  
**subgoal** **by** *(auto simp add: find-Some-iff domIff subset-iff)*  
**subgoal** **by** *(auto simp add: find-Some-iff domIff subset-iff)*  
**subgoal**  
**using**  $\langle \bigwedge x2. [\{0..<card\ S - Suc\ 0\} \subseteq dom\ xors; \bigwedge i. i < card$   
 $S - Suc\ 0 \implies fst\ (the\ (xors\ i)) \subseteq S; find\ (\lambda i. (\lambda\omega. xor\text{-hash } \omega \ xors)$   
 $\in amc.T\ i)\ [Suc\ 0..<card\ S] = Some\ x2] \implies x2 < card\ S \rangle$  **by** *auto*  
**by** *auto*

**from** *xor-hash-family-2-universal[OF S]*  
**have** *univ:prob-space.k-universal (measure-pmf (random-xors S (card*  
 $S - 1)))$  2  
 $xor\text{-hash } \{\alpha. dom\ \alpha = S\} \{\alpha. dom\ \alpha = \{0..<card\ S - 1\}\} .$

**have** *measure-pmf.prob*  
 $(map\text{-pmf } (\lambda s\ w. xor\text{-hash } w\ s)\ (random\text{-xors}\ S\ (card\ S - 1)))$   
 $amc.approxcore\text{-fail} \leq 0.36$   
**apply** *(intro amc.analysis-3[OF - - univ])*  
**apply** *(smt (verit, ccfv-SIG) Groups.mult-ac(3) amc.pivot-def*  
 $assms(1)\ compute\text{-thresh-def more-arith-simps(11)\ real\text{-nat-ceiling-ge})$

```

using  $S$  finite-random-xors-set-pmf apply blast
using  $\varepsilon$  by auto

then have  $b$ : measure-pmf.prob
  (random-xors  $S$  (card  $S - 1$ ))
  {xors.
    (real (( $\lambda(a,b).$   $a*b$ ) (amc.approxcore ( $\lambda\omega.$  xor-hash  $\omega$  xors))))
     $\notin$ 
    {real (card (proj  $S$  (sols  $F$ ))) / ( $1 + \varepsilon$ )..
      ( $1 + \varepsilon$ ) * real (card (proj  $S$  (sols  $F$ )))}}  $\leq 0.36$ 
unfolding amc.approxcore-fail-def
by (auto simp add: case-prod-unfold Let-def)

have  $1$ :  $x \in \text{set-pmf} (\text{random-xors } S (\text{card } S - \text{Suc } 0)) \implies$ 
   $\{0..<\text{card } S - 1\} \subseteq \text{dom } x$  for  $x$ 
by (auto simp add:random-xors-set-pmf[OF S])
have  $2$ :  $x \in \text{set-pmf} (\text{random-xors } S (\text{card } S - \text{Suc } 0)) \implies$ 
   $(\forall i. i < \text{card } S - 1 \longrightarrow \text{fst} (\text{the } (x\ i)) \subseteq S)$  for  $x$ 
proof -
assume  $x$ :  $x \in \text{set-pmf} (\text{random-xors } S (\text{card } S - \text{Suc } 0))$ 
moreover {
  fix  $i$ 
assume  $i$ :  $i < \text{card } S - 1$ 
from random-xors-set-pmf[OF S]
have  $xx$ :  $\text{dom } x = \{..<(\text{card } S - \text{Suc } 0)\}$   $\text{ran } x \subseteq \text{Pow } S \times$ 
UNIV
using  $x$  by auto
obtain  $xi$  where  $xi$ :  $x\ i = \text{Some } xi$ 
using  $xx\ i$  apply clarsimp
by (metis atLeast0LessThan atLeastLessThan-iff bot-nat-0.extremum
domIff option.exhaust surj-pair)
then have  $\text{fst } xi \subseteq S$  using  $xx(2)$ 
unfolding ran-def apply (clarsimp simp add: subset-iff)
by (metis prod.collapse)
then have  $\text{fst} (\text{the } (x\ i)) \subseteq S$  by (simp add: xi)
}
thus  $(\forall i. i < \text{card } S - 1 \longrightarrow \text{fst} (\text{the } (x\ i)) \subseteq S)$  by auto
qed

have measure-pmf.prob (random-xors  $S$  (card  $S - 1$ ))
  {xors. real (approxcore-xors  $F\ S$  thresh xors)  $\in$ 
    {real (card (proj  $S$  (sols  $F$ ))) / ( $1 + \varepsilon$ )..
      ( $1 + \varepsilon$ ) * real (card (proj  $S$  (sols  $F$ )))}} =
measure-pmf.prob
  (random-xors  $S$  (card  $S - 1$ ))
  {xors.
    (real (( $\lambda(a,b).$   $a*b$ ) (amc.approxcore ( $\lambda\omega.$  xor-hash  $\omega$  xors))))
     $\in$ 
    {real (card (proj  $S$  (sols  $F$ ))) / ( $1 + \varepsilon$ )..
```

```

      (1 + ε) * real (card (proj S (sols F))))}}
apply (intro measure-pmf.measure-pmf-eq[where p = (random-xors
S (card S - 1))])
subgoal by auto
apply clarsimp
apply (frule 1)
apply (drule 2)
apply (frule rw)
by auto

moreover have ... =
  1 - measure-pmf.prob
  (random-xors S (card S - 1))
  {xors.
    (real ((λ(a,b). a*b) (amc.approxcore (λω. xor-hash ω xors))))
    ≠
    {real (card (proj S (sols F))) / (1 + ε)..
      (1 + ε) * real (card (proj S (sols F))))}}
apply (subst measure-pmf.prob-compl[symmetric])
by (auto intro!: measure-pmf.measure-pmf-eq)
moreover have ... ≥ 0.64
using b
by auto
ultimately show ?thesis by auto
qed

```

```

lemma compute-t-ge1:
assumes 0 < δ δ < 1
shows compute-t δ n ≥ 1
proof -
have ln (1 / δ) > 0
by (simp add: assms)
then have fix-t: 1 ≤ fix-t δ
unfolding fix-t-def
by (simp add: Suc-leI)

show ?thesis
unfolding compute-t-def
using fix-t
apply (cases n)
unfolding raw-median-bound-def
using assms by auto
qed

```

```

lemma success-arith-bound:
assumes s ≤ (f :: nat)
assumes p ≤ (1::real) q ≤ p 1 / 2 ≤ q
shows p ^ s * (1 - p) ^ f ≤ q ^ s * (1 - q) ^ f
proof -

```

```

have ple:  $p * (1-p) \leq q * (1-q)$ 
  using assms(2-4)
  by (sos ((( $A < 0 * R < 1$ ) + (( $R < 1 * (R < 1 * [p + \sim 1 * q]^2)$ )) +
  (( $A <= 1 * (A <= 2 * R < 1)$ ) * ( $R < 2 * [1]^2$ ))))))

have feq:  $f = (f-s)+s$ 
  using Nat.le-imp-diff-is-add assms(1) by blast
then have  $(1-p)^\wedge f = (1-p)^\wedge s * (1-p)^\wedge (f-s)$ 
  by (metis add commute power-add)
then have  $p^\wedge s * (1-p)^\wedge f =$ 
   $(p * (1-p))^\wedge s * (1-p)^\wedge (f-s)$ 
  by (simp add: power-mult-distrib)
moreover have ...  $\leq$ 
   $(q * (1-q))^\wedge s * (1-p)^\wedge (f-s)$ 
  by (smt (verit) assms(1) assms(2) assms(3) assms(4) diff-self-eq-0
divide-nonneg-nonneg mult-nonneg-nonneg mult-right-mono order-le-less
ple power-0 power-eq-0-iff power-mono zero-less-diff zero-less-power)
moreover have ...  $\leq$ 
   $(q * (1-q))^\wedge s * (1-q)^\wedge (f-s)$ 
  by (smt (verit, ccfv-SIG) assms(2) assms(3) assms(4) divide-eq-0-iff
divide-nonneg-nonneg mult-left-mono mult-pos-pos power-mono zero-less-power)
moreover have ...  $= q^\wedge s * (1-q)^\wedge f$ 
  by (smt (verit) feq mult.assoc mult.commute power-add power-mult-distrib)
ultimately show ?thesis by auto
qed

```

**lemma** *prob-binomial-pmf-upto-mono*:

**assumes**  $1/2 \leq q \leq p \leq 1$

**shows**

*measure-pmf.prob (binomial-pmf n p) {..n div 2}  $\leq$*

*measure-pmf.prob (binomial-pmf n q) {..n div 2}*

**proof** –

**have**  $i: i \leq n \text{ div } 2 \implies$

$p^\wedge i * (1-p)^\wedge (n-i) \leq q^\wedge i * (1-q)^\wedge (n-i)$  **for**  $i$

**using** *assms* **by** (*auto intro!: success-arith-bound*)

**show** ?thesis

**apply** (*subst prob-binomial-pmf-upto*)

**subgoal using** *assms* **by** *auto*

**subgoal using** *assms* **by** *auto*

**apply** (*subst prob-binomial-pmf-upto*)

**subgoal using** *assms* **by** *auto*

**subgoal using** *assms* **by** *auto*

**by** (*auto intro!: sum-mono simp add: i ab-semigroup-mult-class.mult-ac(1)*
*mult-left-mono*)

**qed**

**theorem** *approxmc-sound*:

```

assumes  $\delta$ :  $\delta > 0$   $\delta < 1$ 
assumes  $\varepsilon$ :  $\varepsilon > 0$   $\varepsilon \leq 1$ 
assumes  $S$ : finite  $S$ 
shows measure-pmf.prob (approxmc  $F$   $S$   $\varepsilon$   $\delta$   $n$ )
  {c. real  $c \in$ 
    {real (card (proj  $S$  (sols  $F$ ))) / ( $1 + \varepsilon$ )..
      ( $1 + \varepsilon$ ) * real (card (proj  $S$  (sols  $F$ )))}}
     $\geq 1 - \delta$ 
proof –
define thresh where thresh = compute-thresh  $\varepsilon$ 
define  $t$  where  $t$  = compute-t  $\delta$   $n$ 
then have  $t1$ :  $1 \leq t$ 
  using compute-t-ge1[OF  $\delta$ ] by auto

have card (proj  $S$  (sols  $F$ )) < thresh  $\vee$ 
  card (proj  $S$  (sols  $F$ ))  $\geq$  thresh by auto
moreover {
  assume card (proj  $S$  (sols  $F$ )) < thresh
  then have approxmc  $F$   $S$   $\varepsilon$   $\delta$   $n$  =
    return-pmf (card (proj  $S$  (sols  $F$ )))
  unfolding approxmc-def Let-def thresh-def
  by auto
  then have ?thesis
  unfolding indicator-def of-bool-def
  using assms  $\varepsilon$ 
  by (auto simp add: mult-le-cancel-left1 mult-le-cancel-right1 divide-le-eq)
}
moreover {
  assume  $a$ : card (proj  $S$  (sols  $F$ ))  $\geq$  thresh
  define  $Xf$  where  $Xf = (\lambda(i::nat)$   $xs$ .
    approxcore-xors  $F$   $S$  thresh ( $xs$   $i$ ))

  then have *: approxmc  $F$   $S$   $\varepsilon$   $\delta$   $n$  =
    map-pmf (median  $t$ )
    (prod-pmf { $0..<t$ } ( $\lambda i$ . approxccore  $F$   $S$  thresh))
  using  $a$ 
  unfolding approxmc-def Let-def thresh-def t-def
  by auto

  have **: map-pmf (real  $\circ$  median  $t$ )
    (Pi-pmf { $0..<t$ } (approxcore-xors  $F$   $S$  thresh undefined)
      ( $\lambda i$ . approxccore  $F$   $S$  thresh)) =
    map-pmf ( $\lambda \omega$ . median  $t$  ( $\lambda i$ . real ( $Xf$   $i$   $\omega$ )))
    (prod-pmf { $0..<t$ } ( $\lambda i$ . random-xors  $S$  (card  $S$  - 1)))
  apply (subst median-commute)
  subgoal using  $t1$  by simp
  unfolding approxccore-def
  apply (subst Pi-pmf-map)

```

```

unfolding Xf-def by (auto simp add: pmf.map-comp o-def)

define  $\alpha$  where  $\alpha = (0.14 :: \text{real})$ 
then have  $\alpha: \alpha > 0$  by auto

have indep:prob-space.indep-vars
  (measure-pmf
    (prod-pmf  $\{0..<t\}$  ( $\lambda i. \text{random-xors } S \text{ (card } S - 1)$ )))
  ( $\lambda \cdot. \text{borel}$ ) ( $\lambda x \text{ xa. real (} Xf \text{ x xa)}$ )  $\{0..<t\}$ 
unfolding Xf-def
apply (intro indep-vars-restrict-intro')
by (auto simp add: restrict-dfl-def)

have  $d: \delta \in \{0 < .. < 1\}$  using  $\delta$  by auto

from approxcore-xors-eq[OF thresh-def a  $\in S$ ]
have  $b1: 1 / 2 + \alpha \leq$ 
  measure-pmf.prob (random-xors S (card S - 1))
   $\{xors.$ 
    real (approxcore-xors F S thresh xors)
     $\in \{\text{real (card (proj S (sols F)))} /$ 
       $(1 + \varepsilon) \cdot (1 + \varepsilon) * \text{real (card (proj S (sols F)))}\}$ 
unfolding  $\alpha$ -def by auto
then have  $b2: i < t \implies$ 
   $1 / 2 + \alpha \leq$ 
  measure-pmf.prob
  (prod-pmf  $\{0..<t\}$ 
    ( $\lambda i. \text{random-xors } S \text{ (card } S - 1)$ ))
   $\{\omega \in \text{space}$ 
    (measure-pmf
      (prod-pmf  $\{0..<t\}$ 
        ( $\lambda i. \text{random-xors } S \text{ (card } S - 1)$ )))}.
    real (Xf i  $\omega$ )
     $\in \{\text{real (card (proj S (sols F)))} / (1 + \varepsilon) \cdot$ 
       $(1 + \varepsilon) * \text{real (card (proj S (sols F)))}\}$  for  $i$ 
unfolding Xf-def apply clarsimp
apply (subst prob-prod-pmf-slice)
by auto

have  $***: 1 - \delta$ 
   $\leq \text{measure-pmf.prob (prod-pmf } \{0..<t\} (\lambda i. \text{random-xors } S \text{ (card } S - 1)))$ 
   $\{\omega.$ 
    median t ( $\lambda i. \text{real (} Xf \text{ i } \omega)$ )
     $\in \{\text{real (card (proj S (sols F)))} /$ 
       $(1 + \varepsilon) \cdot (1 + \varepsilon) * \text{real (card (proj S (sols F)))}\}$ 
proof -
have  $t = \text{fix-t } \delta \vee t = n \wedge \text{raw-median-bound } \alpha \ n < \delta$ 
unfolding t-def compute-t-def  $\alpha$ -def by auto

```

```

moreover {
  assume  $t = \text{fix-}t \ \delta$ 
  then have  $tb: -\ln \delta / (2 * \alpha^2) \leq \text{real } t$ 
  unfolding  $\text{fix-}t\text{-def } \alpha\text{-def}$ 
  apply clarsimp
  by (metis assms(1) divide-minus-left inverse-eq-divide ln-inverse
real-nat-ceiling-ge)

  from measure-pmf.median-bound-1[OF  $\alpha$   $d$  indep  $tb$ ]
  have ?thesis using  $b2$  by auto
}
moreover {
  assume  $*$ ;  $t = n$  raw-median-bound  $\alpha$   $n < \delta$ 
  have  $s: 1 / 2 - \alpha = 1 - (1 / 2 + \alpha)$ 
  by auto

  have  $d1: 0 < t$  using  $t1$  by linarith
  have  $d2: 1 / 2 + \alpha \geq 0$  using  $\alpha$  by auto
  have  $d3: \text{interval}\{x..y::\text{real}\}$  for  $x$   $y$ 
  unfolding interval-def by auto

  from prob-space.median-bound-raw[OF -  $d1$   $d2$   $d3$  indep  $b2$ ]
  have  $1 -$ 
    measure-pmf.prob (binomial-pmf  $t$   $(1 / 2 + \alpha)$ )  $\{..t \text{ div } 2\}$ 
     $\leq$  measure-pmf.prob
      (prod-pmf  $\{0..<t\}$ 
        ( $\lambda i.$  random-xors  $S$  ( $\text{card } S - 1$ )))
         $\{\omega.$ 
          median  $t$  ( $\lambda i.$  real ( $Xf$   $i$   $\omega$ ))
           $\in$   $\{\text{real} (\text{card} (\text{proj } S (\text{sols } F))) /$ 
             $(1 + \varepsilon)..(1 + \varepsilon) * \text{real} (\text{card} (\text{proj } S (\text{sols } F)))\}\}$ 
        by (auto simp add: prob-space-measure-pmf)

    moreover have measure-pmf.prob (binomial-pmf  $t$   $(1 / 2 +$ 
 $\alpha)$ )  $\{..t \text{ div } 2\} \leq \delta$ 
    by (smt (verit, ccfv-SIG) * b1 d2 measure-pmf.prob-le-1
prob-binomial-pmf-upto raw-median-bound-def s sum.cong)

  ultimately have ?thesis by auto
}
ultimately show ?thesis by auto
qed

have measure-pmf.prob (approxmc  $F$   $S$   $\varepsilon$   $\delta$   $n$ )
   $\{c.$  real  $c$ 
     $\in$   $\{\text{real} (\text{card} (\text{proj } S (\text{sols } F))) /$ 
       $(1 + \varepsilon)..(1 + \varepsilon) * \text{real} (\text{card} (\text{proj } S (\text{sols } F)))\}\} =$ 
measure-pmf.prob (map-pmf (real  $\circ$  median  $t$ )
    (prod-pmf  $\{0..<t\}$ 

```



```

      (λi. approxmccore F S thresh)))
      {c. c ∈ {real (card (proj S (sols F))) /
        (1 + ε)..(1 + ε) * real (card (proj S (sols F)))}}
using * by auto
moreover have ... =
  measure-pmf.prob (map-pmf (real ∘ median t)
    (map-pmf (λf x. if x ∈ {0..<t} then f x else undefined) (Pi-pmf
      {0..<t} (approxcore-xors F S thresh undefined)
        (λi. approxmccore F S thresh))))
    {c. c ∈ {real (card (proj S (sols F))) /
      (1 + ε)..(1 + ε) * real (card (proj S (sols F)))}}
apply (subst Pi-pmf-default-swap)
by auto
moreover have ... =
  measure-pmf.prob (map-pmf (real ∘ median t)
    (Pi-pmf {0..<t} (approxcore-xors F S thresh undefined)
      (λi. approxmccore F S thresh)))
    {c. c ∈ {real (card (proj S (sols F))) /
      (1 + ε)..(1 + ε) * real (card (proj S (sols F)))}}
by (clarsimp simp add: median-default[symmetric])
moreover have ... ≥ 1 - δ
unfolding **
using ***
by auto
ultimately have ?thesis by auto
}
ultimately show ?thesis by auto
qed

```

To simplify further analyses, we can remove the (required) upper bound on epsilon.

**definition** *mk-eps* :: real ⇒ real  
**where** *mk-eps* ε = (if ε > 1 then 1 else ε)

**definition** *approxm'* ::  
 'fml ⇒ 'a set ⇒  
 real ⇒ real ⇒ nat ⇒ nat pmf  
**where** *approxm'* F S ε δ n =  
*approxm* F S (mk-eps ε) δ n

**corollary** *approxm'-sound*:  
**assumes** δ: δ > 0 δ < 1  
**assumes** ε: ε > 0  
**assumes** S: finite S  
**shows** prob-space.prob (approxm' F S ε δ n)  
 {c. real c ∈  
 {real (card (proj S (sols F))) / (1 + ε)..  
 (1 + ε) \* real (card (proj S (sols F)))}}  
 ≥ 1 - δ

```

proof –
  define  $\varepsilon'$  where  $\varepsilon' = (\text{if } \varepsilon > 1 \text{ then } 1 \text{ else } \varepsilon)$ 
  have  $\varepsilon'$ :  $0 < \varepsilon' \wedge \varepsilon' \leq 1$ 
    unfolding  $\varepsilon'$ -def using  $\varepsilon$  by auto

  from approxmc-sound[OF  $\delta$   $\varepsilon'$  S]
  have *: prob-space.prob (approxmc F S  $\varepsilon'$   $\delta$  n)
    {c. real c  $\in$ 
      {real (card (proj S (sols F))) / ( $1 + \varepsilon'$ )..
        ( $1 + \varepsilon'$ ) * real (card (proj S (sols F)))}}
     $\geq 1 - \delta$  .

  have **:
    {c. real c  $\in$ 
      {real (card (proj S (sols F))) / ( $1 + \varepsilon'$ )..
        ( $1 + \varepsilon'$ ) * real (card (proj S (sols F)))}}
     $\subseteq$ 
    {c. real c  $\in$ 
      {real (card (proj S (sols F))) / ( $1 + \varepsilon$ )..
        ( $1 + \varepsilon$ ) * real (card (proj S (sols F)))}}
    unfolding  $\varepsilon'$ -def
    apply clarsimp
    apply (rule conjI)
    apply (smt (verit) field-sum-of-halves frac-less2 zero-less-divide-iff)
    by (metis (no-types, opaque-lifting) add-le-cancel-left arith-special(3)
      dual-order.trans less-eq-real-def mult-right-mono of-nat-0-le-iff)

  show ?thesis
    apply (intro order.trans[OF *])
    unfolding approxmc'-def  $\varepsilon'$ -def mk-eps-def
    apply (intro measure-pmf.finite-measure-mono)
    using **[unfolded  $\varepsilon'$ -def]
    by auto
qed

```

This shows we can lift all randomness to the top-level (i.e., eagerly sample it).

```

definition approxmc-map::
  'fml  $\Rightarrow$  'a set  $\Rightarrow$ 
  real  $\Rightarrow$  real  $\Rightarrow$  nat  $\Rightarrow$ 
  (nat  $\Rightarrow$  nat  $\Rightarrow$  ('a set  $\times$  bool) option)  $\Rightarrow$ 
  nat
  where approxmc-map F S  $\varepsilon$   $\delta$  n xorsFs = (
    let  $\varepsilon = \text{mk-eps } \varepsilon$  in
    let thresh = compute-thresh  $\varepsilon$  in
    if card (proj S (sols F)) < thresh then
      card (proj S (sols F))
    else
      let t = compute-t  $\delta$  n in

```

$median\ t\ (approxcore\text{-}xors\ F\ S\ thresh\ \circ\ xorsFs)$

**lemma** *approxmc-map-eq*:

**shows**

$map\text{-}pmf\ (approxmc\text{-}map\ F\ S\ \varepsilon\ \delta\ n)$   
 $(Pi\text{-}pmf\ \{0..\lt compute\text{-}t\ \delta\ n\}\ def$   
 $(\lambda i.\ random\text{-}xors\ S\ (card\ S\ -\ 1))) =$   
 $approxmc'\ F\ S\ \varepsilon\ \delta\ n$

**proof** –

**define** *def'* **where**  $def' = approxcore\text{-}xors\ F\ S\ (compute\text{-}thresh$   
 $(mk\text{-}eps\ \varepsilon))\ def$

**have** \*:

$map\text{-}pmf\ (median\ (compute\text{-}t\ \delta\ n))$   
 $(prod\text{-}pmf\ \{0..\lt compute\text{-}t\ \delta\ n\}$   
 $(\lambda i.\ map\text{-}pmf$   
 $(approxcore\text{-}xors\ F\ S$   
 $(compute\text{-}thresh\ (mk\text{-}eps\ \varepsilon)))$   
 $(random\text{-}xors\ S\ (card\ S\ -\ Suc\ 0)))) =$   
 $map\text{-}pmf\ (median\ (compute\text{-}t\ \delta\ n))$   
 $(Pi\text{-}pmf\ \{0..\lt compute\text{-}t\ \delta\ n\}\ def'$   
 $(\lambda i.\ map\text{-}pmf$   
 $(approxcore\text{-}xors\ F\ S$   
 $(compute\text{-}thresh\ (mk\text{-}eps\ \varepsilon)))$   
 $(random\text{-}xors\ S\ (card\ S\ -\ Suc\ 0))))$

**apply** (*subst Pi-pmf-default-swap[symmetric, where dflt = undefined,*  
*where dflt' = def']*)

**by** (*auto simp add: map-pmf-comp median-default[symmetric]*)

**show** *?thesis*

**unfolding** *approxmc'-def approxmc-map-def approxmc-def Let-def*  
*approxmccore-def*

**using** *def'-def*

**by** (*auto simp add: map-pmf-comp Pi-pmf-map[of - - def] \**)

**qed**

**end**

**end**

## 7 Certificate-based ApproxMC

This turns the randomized algorithm into an executable certificate checker

**theory** *CertCheck*

**imports** *ApproxMCAnalysis*

**begin**

## 7.1 ApproxMC with lists instead of sets

**type-synonym**  $'a \text{ xor} = 'a \text{ list} \times \text{bool}$

**definition**  $\text{satisfies-xorL} :: 'a \text{ xor} \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$   
**where**  $\text{satisfies-xorL } xb \ \omega =$   
 $\text{even } (\text{sum-list } (\text{map } (\text{of-bool} \circ \omega) (\text{fst } xb)) +$   
 $\text{of-bool } (\text{snd } xb)::\text{nat})$

**definition**  $\text{sublist-bits} :: 'a \text{ list} \Rightarrow \text{bool list} \Rightarrow 'a \text{ list}$   
**where**  $\text{sublist-bits } ls \ bs =$   
 $\text{map } \text{fst } (\text{filter } \text{snd } (\text{zip } ls \ bs))$

**definition**  $\text{xor-from-bits} ::$   
 $'a \text{ list} \Rightarrow \text{bool list} \times \text{bool} \Rightarrow 'a \text{ xor}$   
**where**  $\text{xor-from-bits } V \ xsb =$   
 $(\text{sublist-bits } V \ (\text{fst } xsb), \ \text{snd } xsb)$

**locale**  $\text{ApproxMCL} =$   
**fixes**  $\text{sols} :: 'fml \Rightarrow ('a \Rightarrow \text{bool}) \text{ set}$   
**fixes**  $\text{enc-xor} :: 'a \text{ xor} \Rightarrow 'fml \Rightarrow 'fml$   
**assumes**  $\text{sols-enc-xor}:$   
 $\bigwedge F \ \text{xor}.$   
 $\text{sols } (\text{enc-xor } \text{xor } F) =$   
 $\text{sols } F \cap \{\omega. \text{satisfies-xorL } \text{xor } \omega\}$

**begin**

**definition**  $\text{list-of-set} :: 'a \text{ set} \Rightarrow 'a \text{ list}$   
**where**  $\text{list-of-set } x = (@ls. \text{set } ls = x \wedge \text{distinct } ls)$

**definition**  $\text{xor-conc} :: 'a \text{ set} \times \text{bool} \Rightarrow 'a \text{ xor}$   
**where**  $\text{xor-conc } xsb = (\text{list-of-set } (\text{fst } xsb), \ \text{snd } xsb)$

**definition**  $\text{enc-xor-conc} :: 'a \text{ set} \times \text{bool} \Rightarrow 'fml \Rightarrow 'fml$   
**where**  $\text{enc-xor-conc} = \text{enc-xor} \circ \text{xor-conc}$

**lemma**  $\text{distinct-count-list}:$   
**assumes**  $\text{distinct } ls$   
**shows**  $\text{count-list } ls \ x = \text{of-bool } (x \in \text{set } ls)$   
**using**  $\text{assms}$   
**apply**  $(\text{induction } ls)$   
**by**  $\text{auto}$

**lemma**  $\text{list-of-set}:$   
**assumes**  $\text{finite } x$   
**shows**  $\text{distinct } (\text{list-of-set } x) \ \text{set } (\text{list-of-set } x) = x$   
**unfolding**  $\text{list-of-set-def}$   
**by**  $(\text{metis } (\text{mono-tags}, \ \text{lifting}) \ \text{assms } \text{finite-distinct-list } \text{someI-ex})+$

**lemma** *count-list-list-of-set*:

**assumes** *finite x*  
**shows** *count-list (list-of-set x) y = of-bool (y ∈ x)*  
**apply** (*subst distinct-count-list*)  
**using** *list-of-set[OF assms]*  
**by** *auto*

**lemma** *satisfies-xorL-xor-conc*:

**assumes** *finite x*  
**shows** *satisfies-xorL (xor-conc (x,b)) ω ↔ satisfies-xor (x,b) {x. ω*  
*x}*  
**unfolding** *satisfies-xorL-def xor-conc-def*  
**using** *list-of-set[OF assms]*  
**by** (*auto simp add: sum-list-map-eq-sum-count count-list-list-of-set[OF*  
*assms] Int-ac(3) assms*)

**sublocale** *appmc: ApproxMC sols enc-xor-conc*

**apply** *unfold-locales*  
**unfolding** *enc-xor-conc-def o-def*  
**apply** (*subst sols-enc-xor*)  
**using** *satisfies-xorL-xor-conc by fastforce*

**definition** *size-xorL* ::

*'fml ⇒ 'a list ⇒*  
*(nat ⇒ bool list × bool) ⇒*  
*nat ⇒ nat*  
**where** *size-xorL F S xorsl i = (*  
*let xors = map (xor-from-bits S ◦ xorsl) [0..*i]* in*  
*let Fenc = fold enc-xor xors F in*  
*card (proj (set S) (sols Fenc)))*

**definition** *check-xorL* ::

*'fml ⇒ 'a list ⇒*  
*nat ⇒*  
*(nat ⇒ bool list × bool) ⇒*  
*nat ⇒ bool*  
**where** *check-xorL F S thresh xorsl i =*  
*(size-xorL F S xorsl i < thresh)*

**definition** *approxcore-xorsL* ::

*'fml ⇒ 'a list ⇒*  
*nat ⇒*  
*(nat ⇒ (bool list × bool)) ⇒*  
*nat*  
**where**  
*approxcore-xorsL F S thresh xorsl =*  
*(case List.find*  
*(check-xorL F S thresh xorsl) [1..*length S*] of*  
*None ⇒ 2 ^ *length S**

| *Some m*  $\Rightarrow$   
 $(2 \wedge m * \text{size-xorL } F \ S \ \text{xorL } m)$

**definition** *xor-abs* :: 'a xor  $\Rightarrow$  'a set  $\times$  bool  
**where** *xor-abs xsb* = (set (fst xsb), snd xsb)

**lemma** *sols-fold-enc-xor*:  
**assumes** *list-all2* ( $\lambda x y.$   
 $\forall w. \text{satisfies-xorL } x \ w = \text{satisfies-xorL } y \ w$ ) *xs ys*  
**assumes** *sols F = sols G*  
**shows** *sols (fold enc-xor xs F) = sols (fold enc-xor ys G)*  
**using** *assms*  
**proof** (*induction xs arbitrary: ys F G*)  
**case** *Nil*  
**then show** ?*case*  
**by** *auto*  
**next**  
**case** (*Cons x xss*)  
**then obtain** *y yss* **where** *ys = y#yss*  
**by** (*meson list-all2-Cons1*)  
**have** *all2*:  $\forall w. \text{satisfies-xorL } x \ w = \text{satisfies-xorL } y \ w$   
**using** *Cons.prem1* *ys* **by** *blast*  
**have** \*: *sols (enc-xor x F) = sols (enc-xor y G)*  
**apply** (*subst sols-enc-xor*)  
**using** *all2 local.Cons(3) sols-enc-xor* **by** *presburger*  
**then show** ?*case* **unfolding** *ys*  
**using** \* *Cons.IH Cons.prem2 local.Cons(2) local.Cons(3) ys* **by**  
*auto*  
**qed**

**lemma** *satisfies-xor-xor-abs*:  
**assumes** *distinct x*  
**show** *satisfies-xor (xor-abs (x,b)) {x.  $\omega$  x}  $\longleftrightarrow$  satisfies-xorL (x,b)*  
 $\omega$   
**unfolding** *satisfies-xorL-def xor-abs-def*  
**apply** (*clarsimp simp add: sum-list-map-eq-sum-count*)  
**by** (*smt (verit, ccfv-SIG) IntD1 Int-commute assms card-eq-sum distinct-count-list of-bool-eq(2) sum.cong*)

**lemma** *xor-conc-xor-abs-rel*:  
**assumes** *distinct (fst x)*  
**show** *satisfies-xorL (xor-conc (xor-abs x)) w  $\longleftrightarrow$*   
 $\text{satisfies-xorL } x \ w$   
**unfolding** *xor-abs-def*  
**apply** (*subst satisfies-xorL-xor-conc*)  
**subgoal** **by** (*simp add: xor-abs-def[symmetric]*)  
**using** *assms satisfies-xor-xor-abs xor-abs-def* **by** *auto*

**lemma** *sorted-sublist-bits*:

**assumes** *sorted V*  
**shows** *sorted (sublist-bits V bs)*  
**unfolding** *sublist-bits-def*  
**using** *assms*  
**by** (*auto intro!: sorted-filter sorted-wrt-take simp add: map-fst-zip-take*)

**lemma** *distinct-sublist-bits:*  
**assumes** *distinct V*  
**shows** *distinct (sublist-bits V bs)*  
**unfolding** *sublist-bits-def*  
**using** *assms*  
**by** (*auto intro!: distinct-map-filter simp add: map-fst-zip-take*)

**lemma** *distinct-fst-xor-from-bits:*  
**assumes** *distinct V*  
**shows** *distinct (fst (xor-from-bits V bs))*  
**unfolding** *xor-from-bits-def*  
**using** *distinct-sublist-bits[OF assms]*  
**by** *auto*

**lemma** *size-xorL:*  
**assumes**  $\bigwedge j. j < i \implies$   
 $xorsf\ j =$   
 $\text{Some } (xor-abs\ (xor-from-bits\ S\ (xorsl\ j)))$   
**assumes** *distinct S*  
**shows** *size-xorL F S xorsl i =*  
 $appmc.size-xor\ F\ (set\ S)\ xorsf\ i$   
**unfolding** *appmc.size-xor-def size-xorL-def*  
**apply** (*clarsimp simp add: enc-xor-conc-def fold-map[symmetric]*)  
**apply** (*intro arg-cong[where f = ( $\lambda x. card\ (proj\ (set\ S)\ x)$ )]*)  
**apply** (*intro sols-fold-enc-xor*)  
**by** (*auto simp add: list-all2-map1 list-all2-map2 list-all2-same assms(1)*  
*assms(2) distinct-fst-xor-from-bits xor-conc-xor-abs-rel*)

**lemma** *fold-enc-xor-more:*  
**assumes**  $x \in sols\ (fold\ enc-xor\ (xs\ @\ rev\ ys)\ F)$   
**shows**  $x \in sols\ (fold\ enc-xor\ xs\ F)$   
**using** *assms*  
**proof** (*induction ys arbitrary: F*)  
**case** *Nil*  
**then show** *?case*  
**by** *auto*  
**next**  
**case** *ih: (Cons y ys)*  
**show** *?case*  
**using** *ih by (auto simp add: ih.prem(1) sols-enc-xor)*  
**qed**

**lemma** *size-xorL-anti-mono:*

**assumes**  $x \leq y$  *distinct S*  
**shows**  $\text{size-xorL } F \ S \ \text{xorsl } x \geq \text{size-xorL } F \ S \ \text{xorsl } y$   
**proof** –  
**obtain**  $ys$  **where**  $ys: [0..<y] = [0..<x] @ \ ys$  *distinct ys*  
**by** (*metis assms(1) bot-nat-0.extremum distinct-upt ordered-cancel-comm-monoid-diff-class.add-diff-in*  
*upt-add-eq-append*)

**define**  $rys$  **where**  $rys = (\text{rev } (\text{map } (\text{xor-from-bits } S \circ \text{xorsl}) \ ys))$   
**have**  $*$ :  $\bigwedge y. y \in \text{set } rys \implies \text{distinct } (\text{fst } y)$   
**unfolding** *rys-def*  
**using** *assms(2) distinct-fst-xor-from-bits*  
**by** (*metis (no-types, opaque-lifting) ex-map-conv o-apply set-rev*)

**show** *?thesis*  
**unfolding** *size-xorL-def Let-def*  
**apply** (*intro card-mono proj-mono*)  
**subgoal using** *card-proj(1)* **by** *blast*  
**unfolding**  $ys$   
**by** (*metis fold-enc-xor-more map-append rev-rev-ident subsetI*)  
**qed**

**lemma** *find-upto-SomeI*:  
**assumes**  $\bigwedge i. a \leq i \implies i < x \implies \neg P \ i$   
**assumes**  $P \ x \ a \leq x \ x < b$   
**shows**  $\text{find } P \ [a..<b] = \text{Some } x$   
**proof** –  
**have**  $x: [a..<b] ! (x-a) = x$   
**by** (*simp add: assms(3) assms(4)*)  
**have**  $j: \bigwedge j. j < x-a \implies \neg P \ ([a..<b] ! j)$   
**using** *assms(1) assms(4)* **by** *auto*  
**show** *?thesis*  
**unfolding** *find-Some-iff*  
**using**  $x \ j$   
**by** (*metis assms(2) assms(3) assms(4) diff-less-mono length-upt*)  
**qed**

**lemma** *check-xorL*:  
**assumes**  $\bigwedge j. j < i \implies$   
 $\text{xorsf } j =$   
 $\text{Some } (\text{xor-abs } (\text{xor-from-bits } S \ (\text{xorsl } j)))$   
**assumes** *distinct S*  
**shows**  $\text{check-xorL } F \ S \ \text{thresh } \text{xorsl } i =$   
 $\text{appmc.check-xor } F \ (\text{set } S) \ \text{thresh } \text{xorsf } i$   
**unfolding** *appmc.check-xor-def check-xorL-def*  
**using** *size-xorL[OF assms]* **by** *presburger*

**lemma** *approxcore-xorsL*:  
**assumes**  $\bigwedge j. j < \text{length } S - 1 \implies$   
 $\text{xorsf } j =$



```

    Some (xor-abs (xor-from-bits S (xorsl j)))
assumes S: distinct S
shows approxcore-xorsL F S thresh xorsl =
    appmc.approxcore-xors F (set S) thresh xorsf
proof –
have c:card (set S) = length S using S
by (simp add: distinct-card)

have *: find (check-xorL F S thresh xorsl) [1..unfolding c
    by (auto intro!: find-cong check-xorL[OF assms(1) S])

have **: find (appmc.check-xor F
    (set S) thresh xorsf)
    [Suc 0..\implies
    j < a  $\implies$ 
    xorsf j =
    Some
    (xor-abs
    (xor-from-bits S
    (xorsl j))) for a j
    unfolding find-Some-iff
    by (auto simp add: assms(1))
show ?thesis
unfolding appmc.approxcore-xors-def approxcore-xorsL-def * c
apply (cases find (appmc.check-xor F (set S) thresh xorsf)
    [Suc 0..subgoal by clarsimp
by (auto intro!: ApproxMCL.size-xorL simp add: ApproxMCL-axioms
    assms **)
qed

```

```

definition approxmc-mapL::
    'fml  $\Rightarrow$  'a list  $\Rightarrow$ 
    real  $\Rightarrow$  real  $\Rightarrow$  nat  $\Rightarrow$ 
    (nat  $\Rightarrow$  nat  $\Rightarrow$  (bool list  $\times$  bool))  $\Rightarrow$ 
    nat
where approxmc-mapL F S  $\varepsilon$   $\delta$  n xorsLs = (
    let  $\varepsilon$  = appmc.mk-eps  $\varepsilon$  in
    let thresh = appmc.compute-thresh  $\varepsilon$  in
    if card (proj (set S) (sols F)) < thresh then
    card (proj (set S) (sols F))
    else
    let t = appmc.compute-t  $\delta$  n in
    median t (approxcore-xorsL F S thresh  $\circ$  xorsLs))

```

```

definition random-xorB :: nat  $\Rightarrow$  (bool list  $\times$  bool) pmf

```

**where** *random-xorB*  $n =$   
*pair-pmf*  
 (*replicate-pmf*  $n$  (*bernoulli-pmf*  $(1/2)$ ))  
 (*bernoulli-pmf*  $(1/2)$ )

**lemma** *approxmc-mapL*:  
**assumes**  $\bigwedge i j. j < \text{length } S - 1 \implies$   
*xorsFs*  $i j =$   
*Some* (*xor-abs* (*xor-from-bits*  $S$  (*xorsLs*  $i j$ )))  
**assumes**  $S$ : *distinct*  $S$   
**shows**  
*approxmc-mapL*  $F S \varepsilon \delta n$  *xorsLs* =  
*appmc.approxmc-map*  $F$  (*set*  $S$ )  $\varepsilon \delta n$  *xorsFs*  
**proof** –  
**show** *?thesis*  
**unfolding** *approxmc-mapL-def* *appmc.approxmc-map-def* *Let-def*  
**using** *assms* **by** (*auto intro!*: *median-cong approxcore-xorsL*)  
**qed**

**lemma** *approxmc-mapL'*:  
**assumes**  $S$ : *distinct*  $S$   
**shows**  
*approxmc-mapL*  $F S \varepsilon \delta n$  *xorsLs* =  
*appmc.approxmc-map*  $F$  (*set*  $S$ )  $\varepsilon \delta n$   
 $(\lambda i j. \text{if } j < \text{length } S - 1$   
*then* *Some* (*xor-abs* (*xor-from-bits*  $S$  (*xorsLs*  $i j$ )))  
*else* *None*)  
**apply** (*intro approxmc-mapL*)  
**using** *assms* **by** *auto*

**lemma** *bits-to-random-xor*:  
**assumes** *distinct*  $S$   
**shows** *map-pmf*  
 $(\lambda x. \text{xor-abs} (\text{xor-from-bits } S x))$   
 $(\text{random-xorB } (\text{length } S)) =$   
 $\text{random-xor } (\text{set } S)$   
**proof** –  
**have**  $\text{xor-abs} (\text{xor-from-bits } S (a,b)) = \text{apfst } (\text{set } \circ \text{sublist-bits } S)$   
 $(a,b)$  **for**  $a b$   
**using** *xor-abs-def* **by** (*auto simp add: xor-from-bits-def*)  
  
**then have**  $*$ :  $(\lambda x. \text{xor-abs} (\text{xor-from-bits } S x)) = \text{apfst } (\text{set } \circ \text{sublist-bits } S)$   
**by** *auto*

**have**  $\bigwedge x. x \in \text{set } S \implies$   
 $z x \implies$   
 $\exists b. (\exists n. S ! n = x \wedge$

```

      map z S ! n = b ∧
      n < length S) ∧ b for z
  by (metis assms distinct-Ex1 nth-map)

then have set (map fst
  (filter snd
    (zip S (map z S)))) =
  {x ∈ set S. Some (z x) = Some True} for z
  by (auto elim: in-set-zipE simp add: in-set-zip image-def )

then have **: map-pmf (set o sublist-bits S)
  (replicate-pmf (length S) (bernoulli-pmf (1 / 2))) =
  map-pmf (λb. {x ∈ set S. b x = Some True})
  (Pi-pmf (set S) (Some undefined)
    (λ-. map-pmf Some (bernoulli-pmf (1 / 2))))
  apply (subst replicate-pmf-Pi-pmf[OF assms, where def = unde-
    fined])
  apply (subst Pi-pmf-map[of - - undefined])
  subgoal by (auto intro!: pmf.map-cong0 simp add: map-pmf-comp
    sublist-bits-def)
  subgoal by (meson set-zip-leftD)
  unfolding map-pmf-comp sublist-bits-def o-def
  by (auto intro!: pmf.map-cong0)

show ?thesis
  unfolding random-xorB-def
  apply (subst random-xor-from-bits)
  by (auto simp add: * ** pair-map-pmf1[symmetric])
qed

lemma Pi-pmf-map-pmf-Some:
  assumes finite S
  shows Pi-pmf S None (λ-. map-pmf Some p) =
    map-pmf (λf v. if v ∈ S then Some (f v) else None)
    (Pi-pmf S def (λ-. p))
proof -
  have *: Pi-pmf S None (λ-. map-pmf Some p) =
    map-pmf (λf x. if x ∈ S then f x else None)
    (Pi-pmf S (Some def) (λ-. map-pmf Some p))
  apply (subst Pi-pmf-default-swap[OF assms])
  by auto

show ?thesis unfolding *
  apply (subst Pi-pmf-map[OF assms, of - def])
  subgoal by simp
  apply (simp add: map-pmf-comp o-def )
  by (metis comp-eq-dest-lhs)
qed

```

**lemma** *bits-to-random-xors*:  
**assumes** *distinct S*  
**shows**  
 $\text{map-pmf } (\lambda f j. \text{if } j < n \text{ then Some } (\text{xor-abs } (\text{xor-from-bits } S (f j))) \text{ else None})$   
 $(\text{Pi-pmf } \{..<(n::\text{nat})\} \text{ def } (\lambda-. \text{random-xorB } (\text{length } S))) =$   
 $\text{random-xors } (\text{set } S) n$   
**unfolding** *random-xors-def*  
**apply** (*subst Pi-pmf-map-pmf-Some*)  
**subgoal using** *assms* **by** *simp*  
**apply** (*subst bits-to-random-xor[symmetric, OF assms]*)  
**apply** (*subst Pi-pmf-map[where dflt = def, where dflt' = xor-abs*  
*(xor-from-bits S def)]*)  
**subgoal by** *simp*  
**subgoal by** *simp*  
**apply** (*clarsimp simp add: map-pmf-comp o-def*)  
**by** (*metis o-apply*)

**lemma** *bits-to-all-random-xors*:  
**assumes** *distinct S*  
**assumes**  $(\lambda j. \text{if } j < n \text{ then Some } (\text{xor-abs } (\text{xor-from-bits } S (\text{def1 } j))) \text{ else None}) = \text{def}$   
**shows**  
 $\text{map-pmf } ((\circ) (\lambda f j. \text{if } j < n \text{ then Some } (\text{xor-abs } (\text{xor-from-bits } S (f j))) \text{ else None}))$   
 $(\text{Pi-pmf } \{0..<(m::\text{nat})\} \text{ def1 } (\lambda-. \text{Pi-pmf } \{..<(n::\text{nat})\} \text{ def2 } (\lambda-. \text{random-xorB } (\text{length } S)))) =$   
 $\text{Pi-pmf } \{0..<m\} \text{ def } (\lambda i. \text{random-xors } (\text{set } S) n)$   
**apply** (*subst bits-to-random-xors[symmetric, OF assms(1), of - def2]*)  
**apply** (*subst Pi-pmf-map[OF -]*)  
**using** *assms(2)* **by** *auto*

**definition** *random-seed-xors::nat  $\Rightarrow$  nat  $\Rightarrow$  (nat  $\Rightarrow$  nat  $\Rightarrow$  bool list  $\times$  bool) pmf*  
**where** *random-seed-xors t l =*  
 $(\text{prod-pmf } \{0..<t\} (\lambda-. \text{prod-pmf } \{..<l-1\} (\lambda-. \text{random-xorB } l)))$

**lemma** *approxmL-sound*:  
**assumes**  $\delta: \delta > 0 \ \delta < 1$

```

assumes  $\varepsilon: \varepsilon > 0$ 
assumes  $S: \text{distinct } S$ 
shows
  prob-space.prob
  (map-pmf (approxmc-mapL  $F S \varepsilon \delta n$ )
    (random-seed-xors (appmc.compute-t  $\delta n$ ) (length  $S$ )))
  {c. real  $c \in$ 
    {real (card (proj (set  $S$ ) (sols  $F$ ))) / ( $1 + \varepsilon$ )..
    ( $1 + \varepsilon$ ) * real (card (proj (set  $S$ ) (sols  $F$ )))}}
   $\geq 1 - \delta$ 
proof -
  define def where def =
    ( $\lambda j. \text{if } j < \text{degree } S$ 
      then Some (xor-abs (xor-from-bits  $S$  (undefined  $j$ )))
      else None)
  have *: (map-pmf (approxmc-mapL  $F S \varepsilon \delta n$ )
    (Pi-pmf { $0..<\text{appmc.compute-t } \delta n$ } undefined
      ( $\lambda-. \text{Pi-pmf } \{..<\text{length } S - 1\}$  undefined
        ( $\lambda-. \text{random-xorB } (\text{length } S)$ )))))) =
    (map-pmf (appmc.approxmc-map  $F$  (set  $S$ )  $\varepsilon \delta n$ )
      (map-pmf (( $\circ$ ) ( $\lambda f.
        (\lambda j. \text{if } j < \text{length } S - 1$ 
          then Some (xor-abs (xor-from-bits  $S$  ( $f j$ )))
          else None)))
        (Pi-pmf { $0..<\text{appmc.compute-t } \delta n$ } undefined
          ( $\lambda-. \text{Pi-pmf } \{..<\text{length } S - 1\}$  undefined
            ( $\lambda-. \text{random-xorB } (\text{length } S)$ )))))))
  unfolding approxmc-mapL'[OF  $S$ ]
  by (simp add: map-pmf-comp o-def)
  have **:
    (map-pmf (approxmc-mapL  $F S \varepsilon \delta n$ )
      (Pi-pmf { $0..<\text{appmc.compute-t } \delta n$ } undefined
        ( $\lambda-. \text{Pi-pmf } \{..<\text{length } S - 1\}$  undefined
          ( $\lambda-. \text{random-xorB } (\text{length } S)$ )))))) =
    map-pmf (appmc.approxmc-map  $F$  (set  $S$ )  $\varepsilon \delta n$ )
      (Pi-pmf { $0..<\text{appmc.compute-t } \delta n$ } def
        ( $\lambda i. \text{random-xors } (\text{set } S) (\text{card } (\text{set } S) - 1)$ )))
  unfolding *
  apply (subst bits-to-all-random-xors[OF  $S$ ])
  using def-def
  by (auto simp add: assms(4) distinct-card)
  show ?thesis
    unfolding ** appmc.approxmc-map-eq random-seed-xors-def
    using  $\delta(2) \varepsilon$  appmc.approxmc'-sound assms(1) by blast
qed

```

```

lemma approxmcL-sound':
  assumes  $\delta: \delta > 0 \delta < 1$ 

```

```

assumes  $\varepsilon: \varepsilon > 0$ 
assumes  $S: \text{distinct } S$ 
shows
  prob-space.prob
    (map-pmf (approxmc-mapL  $F S \varepsilon \delta n$ )
      (random-seed-xors (appmc.compute-t  $\delta n$ ) (length  $S$ )))
    {c. real  $c \notin$ 
      {real (card (proj (set  $S$ ) (sols  $F$ ))) / ( $1 + \varepsilon$ )..
      ( $1 + \varepsilon$ ) * real (card (proj (set  $S$ ) (sols  $F$ )))}}  $\leq \delta$ 
proof –
  have prob-space.prob
    (map-pmf (approxmc-mapL  $F S \varepsilon \delta n$ )
      (Pi-pmf { $0..<$ appmc.compute-t  $\delta n$ } undefined
        ( $\lambda$ -. Pi-pmf { $..<$ length  $S - 1$ } undefined
          ( $\lambda$ -. random-xorB (length  $S$ ))))))
    {c. real  $c \notin$ 
      {real (card (proj (set  $S$ ) (sols  $F$ ))) / ( $1 + \varepsilon$ )..
      ( $1 + \varepsilon$ ) * real (card (proj (set  $S$ ) (sols  $F$ )))}} =
  1 – prob-space.prob
    (map-pmf (approxmc-mapL  $F S \varepsilon \delta n$ )
      (Pi-pmf { $0..<$ appmc.compute-t  $\delta n$ } undefined
        ( $\lambda$ -. Pi-pmf { $..<$ length  $S - 1$ } undefined
          ( $\lambda$ -. random-xorB (length  $S$ ))))))
    {c. real  $c \in$ 
      {real (card (proj (set  $S$ ) (sols  $F$ ))) / ( $1 + \varepsilon$ )..
      ( $1 + \varepsilon$ ) * real (card (proj (set  $S$ ) (sols  $F$ )))}}
  apply (subst measure-pmf.prob-compl[symmetric])
  by (auto simp add: vimage-def)
  thus ?thesis using approxmcL-sound[OF assms, of  $F n$ ]  $\delta$ 
  unfolding random-seed-xors-def
  by linarith
qed

end

```

## 7.2 ApproxMC certificate checker

```

definition str-of-bool :: bool  $\Rightarrow$  String.literal
  where str-of-bool  $b =$  (
    if  $b$  then STR "true" else STR "false")

fun str-of-nat-aux :: nat  $\Rightarrow$  char list  $\Rightarrow$  char list
  where str-of-nat-aux  $n$  acc = (
    let  $c =$  char-of-integer (of-nat ( $48 + n \bmod 10$ )) in
    if  $n < 10$  then  $c \#$  acc
    else str-of-nat-aux ( $n \text{ div } 10$ ) ( $c \#$  acc))

definition str-of-nat :: nat  $\Rightarrow$  String.literal
  where str-of-nat  $n =$  String.implode (str-of-nat-aux  $n$  [])

```

**type-synonym**  $'a \text{ sol} = ('a \times \text{bool}) \text{ list}$

**definition**  $\text{canon-map-of} :: ('a \times \text{bool}) \text{ list} \Rightarrow ('a \Rightarrow \text{bool})$   
**where**  $\text{canon-map-of } ls =$   
     $(\text{let } m = \text{map-of } ls \text{ in}$   
     $(\lambda x. \text{case } m \text{ } x \text{ of } \text{None} \Rightarrow \text{False} \mid \text{Some } b \Rightarrow b))$

**lemma**  $\text{canon-map-of}[code]:$   
**shows**  $\text{canon-map-of } ls =$   
     $(\text{let } m = \text{Mapping.of-alist } ls \text{ in}$   
     $\text{Mapping.lookup-default } \text{False } m)$   
**unfolding**  $\text{canon-map-of-def lookup-default-def}$   
**by**  $(\text{auto simp add: lookup-of-alist})$

**definition**  $\text{proj-sol} :: 'a \text{ list} \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool list}$   
**where**  $\text{proj-sol } S \ w = \text{map } w \ S$

The following extended locale assumes additional support for syntactically working with solutions

**locale**  $\text{CertCheck} = \text{ApproxMCL } \text{sols } \text{enc-xor}$   
**for**  $\text{sols} :: 'fml \Rightarrow ('a \Rightarrow \text{bool}) \text{ set}$   
**and**  $\text{enc-xor} :: 'a \text{ list} \times \text{bool} \Rightarrow 'fml \Rightarrow 'fml +$   
**fixes**  $\text{check-sol} :: 'fml \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$   
**fixes**  $\text{ban-sol} :: 'a \text{ sol} \Rightarrow 'fml \Rightarrow 'fml$   
**assumes**  $\text{sols-ban-sol}:$   
     $\bigwedge F \text{ vs.}$   
     $\text{sols } (\text{ban-sol } \text{vs } F) =$   
     $\text{sols } F \cap \{\omega. \text{map } \omega \ (\text{map } \text{fst } \text{vs}) \neq \text{map } \text{snd } \text{vs}\}$   
**assumes**  $\text{check-sol}:$   
     $\bigwedge F \ w. \text{check-sol } F \ w \longleftrightarrow w \in \text{sols } F$   
**begin**

Assuming parameter access to an UNSAT checking oracle

**context**  
**fixes**  $\text{check-unsat} :: 'fml \Rightarrow \text{bool}$   
**begin**

Throughout this checker, INL indicates error, INR indicates success

**definition**  $\text{check-BSAT-sols}::$   
     $'fml \Rightarrow 'a \text{ list} \Rightarrow \text{nat} \Rightarrow ('a \Rightarrow \text{bool}) \text{ list} \Rightarrow \text{String.literal} + \text{unit}$   
**where**  $\text{check-BSAT-sols } F \ S \ \text{thresh } \text{cms} =$   
     $\text{let } ps = \text{map } (\text{proj-sol } S) \ \text{cms} \text{ in}$   
     $\text{let } b1 = \text{list-all } (\text{check-sol } F) \ \text{cms} \text{ in}$   
     $\text{let } b2 = \text{distinct } ps \text{ in}$   
     $\text{let } b3 =$

```

(length cms < thresh →
  check-unsat (fold ban-sol (map (zip S) ps) F)) in
if b1 ∧ b2 ∧ b3 then Inr ()
else Inl (STR "checks ---" +
  STR " all valid sols: " + str-of-bool b1 +
  STR ", all distinct sols: " + str-of-bool b2 +
  STR ", unsat check (< thresh sols): " + str-of-bool b3)
)

```

**definition** *BSAT* ::  
'*fml* ⇒ '*a* list ⇒ nat ⇒ ('*a* ⇒ bool) list ⇒ String.literal + nat  
**where** *BSAT* *F S thresh xs* = (  
 case check-BSAT-sols *F S thresh xs* of  
 | Inl *err* ⇒ Inl *err*  
 | Inr - ⇒ Inr (length *xs*)  
)

**definition** *size-xorL-cert* ::  
'*fml* ⇒ '*a* list ⇒ nat ⇒  
(nat ⇒ (bool list × bool)) ⇒ nat ⇒  
('*a* ⇒ bool) list ⇒ String.literal + nat  
**where** *size-xorL-cert* *F S thresh xorsl i xs* = (  
 let *xors* = map (xor-from-bits *S* ∘ *xorsl*) [0..<*i*] in  
 let *Fenc* = fold enc-xor *xors F* in  
*BSAT* *Fenc S thresh xs*  
)

**fun** *approxcore-xorsL-cert* ::  
'*fml* ⇒ '*a* list ⇒ nat ⇒  
nat × ('*a* ⇒ bool) list × ('*a* ⇒ bool) list ⇒  
(nat ⇒ (bool list × bool))  
⇒ String.literal + nat  
**where** *approxcore-xorsL-cert* *F S thresh (m,cert1,cert2) xorsl* = (  
 if 1 ≤ *m* ∧ *m* ≤ length *S*  
 then  
 case *size-xorL-cert* *F S thresh xorsl (m-1) cert1* of  
 | Inl *err* ⇒ Inl (STR "cert1 " + *err*)  
 | Inr *n* ⇒  
 if *n* ≥ *thresh*  
 then  
 if *m* = length *S*  
 then Inr (2 ^ length *S*)  
 else  
 case *size-xorL-cert* *F S thresh xorsl m cert2* of  
 | Inl *err* ⇒ Inl (STR "cert2 " + *err*)  
 | Inr *c* ⇒  
 if *c* < *thresh* then Inr (2 ^ *m* \* *c*)  
)



```

    else Inl (STR "too many solutions at m added XORs")
  else Inl (STR "too few solutions at m-1 added XORs")
else
  Inl (STR "invalid value of m, need 1 <= m <= |S|")

```

**definition** *find-t* :: real ⇒ nat

```

where find-t δ = (
  case find (λi. appmc.raw-median-bound 0.14 i < δ) [0..<256] of
    Some m ⇒ m
  | None ⇒ appmc.fix-t δ
)

```

**fun** *fold-approxcore-xorsL-cert*::

```

'fml ⇒ 'a list ⇒ nat ⇒
nat ⇒ nat ⇒
(nat ⇒ (nat × ('a ⇒ bool) list × ('a ⇒ bool) list)) ⇒
(nat ⇒ nat ⇒ (bool list × bool))
⇒ String.literal + (nat list)
where
  fold-approxcore-xorsL-cert F S thresh t 0 cs xorsLs = Inr []
| fold-approxcore-xorsL-cert F S thresh t (Suc i) cs xorsLs = (
  let it = t - Suc i in
  case approxcore-xorsL-cert F S thresh (cs it) (xorsLs it) of
    Inl err ⇒ Inl (STR "round " + str-of-nat it + STR " " + err)
  | Inr n ⇒
    (case fold-approxcore-xorsL-cert F S thresh t i cs xorsLs of
      Inl err ⇒ Inl err
    | Inr ns ⇒ Inr (n # ns)))
)

```

**definition** *calc-median*::

```

'fml ⇒ 'a list ⇒ nat ⇒ nat ⇒
(nat ⇒ (nat × ('a ⇒ bool) list × ('a ⇒ bool) list)) ⇒
(nat ⇒ nat ⇒ (bool list × bool)) ⇒
String.literal + nat
where calc-median F S thresh t ms xorsLs = (
  case fold-approxcore-xorsL-cert F S thresh t t ms xorsLs of
    Inl err ⇒ Inl err
  | Inr ls ⇒ Inr (sort ls ! (t div 2))
)

```

**fun** *certcheck*::

```

'fml ⇒ 'a list ⇒
real ⇒ real ⇒
(('a ⇒ bool) list ×
(nat ⇒ (nat × ('a ⇒ bool) list × ('a ⇒ bool) list))) ⇒
(nat ⇒ nat ⇒ (bool list × bool)) ⇒
String.literal + nat
where certcheck F S ε δ (m0,ms) xorsLs = (

```

```

let  $\varepsilon = \text{appmc.mk-eps } \varepsilon$  in
let  $\text{thresh} = \text{appmc.compute-thresh } \varepsilon$  in
case  $\text{BSAT } F \ S \ \text{thresh } m0$  of  $\text{Inl err} \Rightarrow \text{Inl err}$ 
|  $\text{Inr } Y \Rightarrow$ 
  if  $Y < \text{thresh}$  then  $\text{Inr } Y$ 
  else
    let  $t = \text{find-t } \delta$  in
    calc-median  $F \ S \ \text{thresh } t \ \text{ms } \text{xorsLs}$ 

```

**context**

```

assumes  $\text{check-unsat}: \bigwedge F. \text{check-unsat } F \Longrightarrow \text{sols } F = \{\}$ 
begin

```

**lemma** *sols-fold-ban-sol*:

```

shows  $\text{sols } (\text{fold ban-sol } ls \ F) =$ 
 $\text{sols } F \cap \{\omega. (\forall vs \in \text{set } ls. \text{map } \omega \ (\text{map } \text{fst } vs) \neq \text{map } \text{snd } vs)\}$ 
proof (induction  $ls$  arbitrary:  $F$ )
  case Nil
  then show ?case by auto
next
  case (Cons  $vs \ ls$ )
  show ?case
    using  $\text{Cons}(1)$  sols-ban-sol
    by auto
qed

```

**lemma** *inter-cong-right*:

```

assumes  $\bigwedge x. x \in A \Longrightarrow x \in B \longleftrightarrow x \in C$ 
shows  $A \cap B = A \cap C$ 
using assms by auto

```

**lemma** *proj-sol-canon-map-of*:

```

assumes distinct  $S$   $\text{length } S = \text{length } w$ 
shows  $\text{proj-sol } S \ (\text{canon-map-of } (\text{zip } S \ w)) = w$ 
using assms
unfolding proj-sol-def canon-map-of-def
proof (induction  $w$  arbitrary:  $S$ )
  case Nil
  then show ?case
    by auto
next
  case (Cons  $a \ w$ )
  obtain  $s \ ss$  where  $S = s \ \# \ ss$ 
  by (metis  $\text{Cons.prem}(2)$  Suc-le-length-iff order.refl)
  then show ?case
    apply clarsimp
    by (smt ( $z3$ )  $\text{Cons.IH } \text{Cons.prem}(2)$  add.right-neutral add-Suc-right
distinct.simps}(2) list.size}(4) local.Cons}(2) map-eq-conv nat.inject)

```

qed

**lemma** *proj-sol-cong*:

**assumes** *restr* (set *S*) *A* = *restr* (set *S*) *B*  
**shows** *proj-sol* *S* *A* = *proj-sol* *S* *B*  
**using** *assms*  
**unfolding** *proj-sol-def restr-def map-eq-conv*  
**by** (*metis option.simps(1)*)

**lemma** *canon-map-of-map-of*:

**assumes** *length* *S* = *length* *x*  
**assumes** *canon-map-of* (zip *S* *x*) ∈ *A*  
**shows** *map-of* (zip *S* *x*) ∈ *proj* (set *S*) *A*  
**proof** –  
**define** *f* **where** *f* = (λ*xa*. *case map-of* (zip *S* *x*) *xa* of  
    *None* ⇒ *False* | *Some* *b* ⇒ *b*)  
**have** *map-of* (zip *S* *x*) =  
    (λ*y*. *if* *y* ∈ set *S* then *Some* (*f* *y*) else *None*)  
**unfolding** *f-def fun-eq-iff*  
**using** *map-of-zip-is-Some[OF assms(1)]*  
**by** (*metis option.case-eq-if option.distinct(1) option.exhaust option.sel*)  
**thus** *?thesis*  
**using** *assms* **unfolding** *canon-map-of-def ApproxMCCore.proj-def restr-def image-def*  
**using** *f-def* **by** *auto*  
qed

**lemma** *proj-proj-sol-map-of-zip-1*:

**assumes** *distinct* *S* *length* *S* = *length* *w*  
**assumes** *w*: *w* ∈ *rdb*  
**shows**  
    *map-of* (zip *S* *w*) ∈  
    *proj* (set *S*) {ω. *proj-sol* *S* ω ∈ *rdb*}  
**apply** (*intro canon-map-of-map-of[OF assms(2)]*)  
**using** *proj-sol-canon-map-of[OF assms(1–2)]* *w* **by** *auto*

**lemma** *proj-proj-sol-map-of-zip-2*:

**assumes**  $\bigwedge bs. bs \in rdb \implies length\ bs = length\ S$   
**assumes** *w*: *w* ∈ *proj* (set *S*) {ω. *proj-sol* *S* ω ∈ *rdb*}  
**shows**  
    *w* ∈ (*map-of* ∘ *zip* *S*) ‘ *rdb*

**proof** –

**obtain** *ww* **where** *ww*: *proj-sol* *S* *ww* ∈ *rdb* *w* = *restr* (set *S*) *ww*  
**using** *w* **unfolding** *ApproxMCCore.proj-def*  
**by** *auto*

**have** *w* = *map-of* (zip *S* (*proj-sol* *S* *ww*))  
**unfolding** *ww restr-def proj-sol-def map-of-zip-map*

```

    by auto

  thus ?thesis using ww
  by (auto simp add: image-def)
qed

lemma proj-proj-sol-map-of-zip:
  assumes distinct S
  assumes  $\bigwedge bs. bs \in rdb \implies \text{length } bs = \text{length } S$ 
  shows
    proj (set S) { $\omega. \text{proj-sol } S \ \omega \in rdb$ } =
    (map-of  $\circ$  zip S) ' rdb
  apply (rule antisym)
  subgoal
    using proj-proj-sol-map-of-zip-2[OF assms(2)]
    by blast
  using assms(2)
  by (auto intro!: proj-proj-sol-map-of-zip-1[OF assms(1)])

definition ban-proj-sol :: 'a list  $\Rightarrow$  ('a  $\Rightarrow$  bool) list  $\Rightarrow$  'fml  $\Rightarrow$  'fml
where ban-proj-sol S xs F =
  fold ban-sol (map (zip S  $\circ$  proj-sol S) xs) F

lemma check-sol-imp-proj:
  assumes w  $\in$  sols F
  shows map-of (zip S (proj-sol S w))  $\in$  proj (set S) (sols F)
  unfolding proj-sol-def map-of-zip-map ApproxMCCore.proj-def im-
age-def restr-def
  using assms by auto

lemma checked-BSAT-lower:
  assumes S: distinct S
  assumes check-BSAT-sols F S thresh xs = Inr ()
  shows length xs  $\leq$  card (proj (set S) (sols F))
  length xs < thresh  $\implies$ 
    card (proj (set S) (sols F)) = length xs
proof -
  define Sxs where Sxs = map (proj-sol S) xs
  have dSxs: distinct Sxs
    using assms unfolding Sxs-def check-BSAT-sols-def Let-def
    by (auto split: if-splits)

  have lSxs:  $\bigwedge x. x \in \text{set } Sxs \implies \text{length } x = \text{length } S$ 
    unfolding Sxs-def proj-sol-def by auto
  define SSxs where SSxs = map (zip S) Sxs
  have dSSxs: distinct (map map-of SSxs)
    unfolding SSxs-def
    using dSxs unfolding inj-on-def distinct-map
    by (smt (verit) assms(1) imageE lSxs list.set-map map-of-zip-inject)

```

```

have *: set (map map-of SSxs)  $\subseteq$  proj (set S) (sols F)
  unfolding Sxs-def SSxs-def
  using assms unfolding check-BSAT-sols-def Let-def
  by (auto intro!: check-sol-imp-proj-split: if-splits simp add: check-sol
list-all-iff)
  have length xs = card (set (map map-of SSxs))
  by (metis SSxs-def Sxs-def dSSxs length-map length-remdups-card-conv
remdups-id-iff-distinct)

thus length xs  $\leq$  card (proj (set S) (sols F))
  by (metis * List.finite-set card-mono card-proj(1))

have frr1: ( $\forall$  vs  $\in$  set SSxs. map  $\omega$  (map fst vs)  $\neq$  map snd vs)  $\implies$ 
  ( $\forall$  vs  $\in$  set Sxs. proj-sol S  $\omega \neq$  vs) for  $\omega$ 
  apply (clarsimp simp add: proj-sol-def SSxs-def)
  by (metis (mono-tags, lifting) in-set-zip nth-map)
have frr2: ( $\forall$  vs  $\in$  set Sxs. proj-sol S  $\omega \neq$  vs)  $\implies$ 
  (vs  $\in$  set SSxs  $\implies$  map  $\omega$  (map fst vs)  $\neq$  map snd vs) for vs  $\omega$ 
  apply (clarsimp simp add: proj-sol-def SSxs-def)
  by (smt (z3) Sxs-def assms(1) length-map length-map length-map
map-eq-conv map-fst-zip map-of-zip-inject mem-Collect-eq nth-map nth-map
nth-map proj-sol-def set-conv-nth set-conv-nth zip-map-fst-snd)

have frr:  $\{\omega. (\forall$  vs  $\in$  set SSxs. map  $\omega$  (map fst vs)  $\neq$  map snd vs)\}
=
   $\{\omega. (\forall$  vs  $\in$  set Sxs. proj-sol S  $\omega \neq$  vs)\}
  using frr1 frr2 by auto

moreover {
  assume length xs < thresh
  then have sols (ban-proj-sol S xs F) = {}
  apply (intro check-unsat)
  using assms(2) unfolding check-BSAT-sols-def Let-def
  by (auto simp add: ban-proj-sol-def o-assoc split: if-splits)

  then have sols F  $\cap$   $\{\omega. (\forall$  vs  $\in$  set Sxs. proj-sol S  $\omega \neq$  vs)\} = {}
  unfolding ban-proj-sol-def sols-fold-ban-sol
  frr[symmetric]
  by (auto simp add: SSxs-def Sxs-def)
  then have 1:proj (set S) (sols F)  $\cap$ 
proj (set S)
   $\{\omega. \forall$  vs  $\in$  set Sxs. proj-sol S  $\omega \neq$  vs\} = {}
  unfolding ApproxMCCore.proj-def
  using proj-sol-cong
  by (smt (verit, del-insts) disjoint-iff-not-equal image-iff mem-Collect-eq)

  have 2: proj (set S) (sols F)  $\cap$   $\neg$ proj (set S)  $\{\omega. (\forall$  vs  $\in$  set Sxs.
proj-sol S  $\omega \neq$  vs)\} =

```

$\text{proj } (\text{set } S) (\text{sols } F) \cap \text{proj } (\text{set } S) \{ \omega. \text{proj-sol } S \omega \in \text{set } Sxs \}$   
**apply** (*intro inter-cong-right*)  
**by** (*auto intro!: proj-sol-cong simp add: ApproxMCCore.proj-def*)

**have** 3:  $\text{proj } (\text{set } S) \{ \omega. \text{proj-sol } S \omega \in \text{set } Sxs \} = (\text{map-of} \circ \text{zip } S) \text{ ` } (\text{set } Sxs)$   
**apply** (*intro proj-proj-sol-map-of-zip[OF S]*)  
**using** *lSxs by auto*

**have** 4:  $\text{proj } (\text{set } S) (\text{sols } F) \cap (\text{map-of} \circ \text{zip } S) \text{ ` } (\text{set } Sxs) =$   
 $(\text{map-of} \circ \text{zip } S) \text{ ` } (\text{set } Sxs)$   
**using** \* *SSxs-def by auto*

**have** \*\*:  $\text{proj } (\text{set } S) (\text{sols } F) =$   
 $\text{proj } (\text{set } S) (\text{sols } F) \cap \text{proj } (\text{set } S) \{ \omega. (\forall vs \in \text{set } Sxs. \text{proj-sol } S \omega \neq vs) \} \cup$   
 $\text{proj } (\text{set } S) (\text{sols } F) \cap \neg \text{proj } (\text{set } S) \{ \omega. (\forall vs \in \text{set } Sxs. \text{proj-sol } S \omega \neq vs) \}$   
**by** *auto*

**have** *card* ( $\text{proj } (\text{set } S) (\text{sols } F)$ ) =  
 $\text{card } ((\text{map-of} \circ \text{zip } S) \text{ ` } (\text{set } Sxs))$   
**apply** (*subst \*\**)  
**apply** (*subst card-Un-disjoint*)  
**using** 1 2 3 4 **by** (*auto simp add: card-proj(1)*)

**then have** *card* ( $\text{proj } (\text{set } S) (\text{sols } F)$ ) = *length xs*  
**by** (*simp add: SSxs-def <length xs = card (set (map map-of SSxs))>*)  
**thus**  $\text{length } xs < \text{thresh} \implies \text{card } (\text{proj } (\text{set } S) (\text{sols } F)) = \text{length } xs$   
**by** *auto*  
**qed**

**lemma** *good-BSAT*:

**assumes** *distinct S*  
**assumes** *BSAT F S thresh xs = Inr n*  
**shows**  $n \leq \text{card } (\text{proj } (\text{set } S) (\text{sols } F))$   
 $n < \text{thresh} \implies$   
 $\text{card } (\text{proj } (\text{set } S) (\text{sols } F)) = n$   
**using** *checked-BSAT-lower[OF assms(1)] assms(2)*  
**by** (*auto simp add: BSAT-def split: if-splits sum.splits*)

**lemma** *size-xorL-cert*:

**assumes** *distinct S*  
**assumes** *size-xorL-cert F S thresh xorsl i xs = Inr n*  
**shows**  
 $\text{size-xorL } F S \text{ xorsl } i \geq n$

```

     $n < thresh \longrightarrow size\text{-}xorL\ F\ S\ xorSl\ i = n$ 
using assms unfolding size-xorL-def size-xorL-cert-def
using good-BSAT by auto

lemma approxcore-xorsL-cert:
assumes S: distinct S
assumes approxcore-xorsL-cert F S thresh mc xorSl = Inr n
shows approxcore-xorsL F S thresh xorSl = n
proof –
obtain m cert1 cert2 where mc: mc = (m, cert1, cert2)
using prod-cases3 by blast
obtain nn1 where
nn1: size-xorL-cert F S thresh xorSl (m-1) cert1 = Inr nn1
using assms unfolding mc
by (auto split: if-splits sum.splits)

from size-xorL-cert[OF S this]
have nn1:
nn1 ≤ size-xorL F S xorSl (m - 1)
nn1 < thresh → size-xorL F S xorSl (m - 1) = nn1 by auto

have m:  $1 \leq m$   $m \leq length\ S$ 
nn1 ≥ thresh and
ms: m = length S ∧ n = 2 ^ length S ∨
m < length S
using nn1 assms unfolding mc
by (auto split: if-splits simp add: Let-def)

have bnd:  $\bigwedge i. 1 \leq i \implies i \leq m - 1 \implies$ 
size-xorL F S xorSl i ≥ thresh
using nn1 m(3)
by (meson assms(1) dual-order.trans size-xorL-anti-mono)

note ms
moreover {
assume *:  $m = length\ S$ 
then have find (check-xorL F S thresh xorSl) [1..<length S] =
None
unfolding find-None-iff check-xorL-def
by (auto simp add: bnd leD)
then have ?thesis
unfolding approxcore-xorsL-def
using * ms by force
}
moreover {
assume *:  $m < length\ S$ 

obtain nn2 where
nn2: size-xorL-cert F S thresh xorSl m cert2 = Inr nn2

```

```

      nn2 < thresh
      n = 2 ^ m * nn2
      using assms * unfolding mc
      by (auto split: if-splits sum.splits)

    from size-xorL-cert[OF S nn2(1)]
    have nn2l:
      size-xorL F S xorsl m = nn2
      using nn2(2) by blast

    then have find (check-xorL F S thresh xorsl) [Suc 0..<length S]
    = Some m
      apply (intro find-upto-SomeI)
      subgoal by (auto simp add: m * bnd check-xorL-def leD)
      subgoal
        using * calculation(1) size-xorL-cert(2) nn2
        by (auto simp add: m * bnd check-xorL-def leD)
      subgoal using m(1) by linarith
      by (auto simp add: m * bnd check-xorL-def leD)

    then have ?thesis
      unfolding approxcore-xorsL-def using nn2 nn2l
      by auto
  }
  ultimately show ?thesis by auto
qed

lemma fold-approxcore-xorsL-cert:
  assumes S: distinct S
  assumes i ≤ t
  assumes fold-approxcore-xorsL-cert F S thresh t i cs xorsLs = Inr
  ns
  shows map (approxcore-xorsL F S thresh ∘ xorsLs) [t-i..<t] = ns
  using assms(2-3)
proof (induction i arbitrary: ns)
  case 0
  then show ?case
    by auto
next
  case c:(Suc i)
  from c(3)
  obtain n nss where *:ns = n # nss
    fold-approxcore-xorsL-cert F S thresh t i cs xorsLs = Inr nss
    approxcore-xorsL-cert F S thresh (cs (t-Suc i)) (xorsLs (t-Suc
  i)) = Inr n
  by (auto simp add: Let-def split: sum.splits)

  have i ≤ t using c by auto
  from c(1)[OF this *(2)]

```



```

have 2:  $nss =$ 
   $map (approxcore-xorsL F S thresh \circ xorsLs) [t - i..<t]$ 
  by auto

have  $i:[t - Suc i..<t] = (t - Suc i) \# [t - i..<t]$ 
  apply (subst upt-rec)
  using  $c(2)$ 
  using Suc-diff-Suc Suc-le-lessD by presburger

show ?case
  unfolding  $i *$ 
  apply (clarsimp simp add: 2)
  using  $2 *(3) approxcore-xorsL-cert assms(1)$  by blast
qed

lemma calc-median:
  assumes  $S: distinct S$ 
  assumes calc-median  $F S thresh t ms xorsLs = Inr n$ 
  shows median  $t (approxcore-xorsL F S thresh \circ xorsLs) = n$ 
  using assms
  unfolding calc-median-def median-def
  apply (clarsimp simp add: assms split: if-splits sum.splits)
  using fold-approxcore-xorsL-cert[OF S]
  by (metis diff-is-0-eq' dual-order.refl)

lemma compute-t-find-t[simp]:
  shows appmc.compute-t  $\delta (find-t \delta) = find-t \delta$ 
  unfolding find-t-def appmc.compute-t-def
  apply (clarsimp simp add: option.case-eq-if)
  unfolding find-Some-iff
  by auto

lemma certcheck:
  assumes  $distinct S$ 
  assumes certcheck  $F S \varepsilon \delta (m0,ms) xorsLs = Inr n$ 
  shows approxmc-mapL  $F S \varepsilon \delta (find-t \delta) xorsLs = n$ 
  using assms(2)
  unfolding approxmc-mapL-def
  using good-BSAT apply (clarsimp split: sum.splits if-splits simp
add: Let-def)
  subgoal using order-le-less-trans assms by blast
  using assms order.strict-trans1
  by (meson assms(1) calc-median)

lemma certcheck':
  assumes  $distinct S$ 
  assumes  $\neg isl (certcheck F S \varepsilon \delta m xorsLs)$ 
  shows projr (certcheck  $F S \varepsilon \delta m xorsLs$ ) =
    approxmc-mapL  $F S \varepsilon \delta (find-t \delta) xorsLs$ 

```

by (metis certcheck assms(1) assms(2) sum.collapse(2) surj-pair)

**lemma** *certcheck-sound*:

**assumes**  $\delta$ :  $\delta > 0$   $\delta < 1$

**assumes**  $\varepsilon$ :  $\varepsilon > 0$

**assumes**  $S$ : *distinct*  $S$

**shows**

*measure-pmf.prov*  
 (map-pmf ( $\lambda r$ . certcheck  $F S \varepsilon \delta$  ( $f r$ )  $r$ )  
 (random-seed-xors (find-t  $\delta$ ) (length  $S$ )))  
 { $c$ .  $\neg$ isl  $c \wedge$   
 real (projr  $c$ )  $\notin$   
 {real (card (proj (set  $S$ ) (sols  $F$ ))) / (1 +  $\varepsilon$ )..  
 (1 +  $\varepsilon$ ) \* real (card (proj (set  $S$ ) (sols  $F$ )))}}  $\leq \delta$

**proof** –

**have** *measure-pmf.prov*

(map-pmf ( $\lambda r$ . certcheck  $F S \varepsilon \delta$  ( $f r$ )  $r$ )  
 (random-seed-xors (find-t  $\delta$ ) (length  $S$ )))  
 { $c$ .  $\neg$ isl  $c \wedge$   
 real (projr  $c$ )  $\notin$   
 {real (card (proj (set  $S$ ) (sols  $F$ ))) / (1 +  $\varepsilon$ )..  
 (1 +  $\varepsilon$ ) \* real (card (proj (set  $S$ ) (sols  $F$ )))}}  $\leq$

*prob-space.prov*

(map-pmf (approxmc-mapL  $F S \varepsilon \delta$  (find-t  $\delta$ ))  
 (random-seed-xors (appmc.compute-t  $\delta$  (find-t  $\delta$ )) (length  $S$ )))  
 { $c$ . real  $c \notin$   
 {real (card (proj (set  $S$ ) (sols  $F$ ))) / (1 +  $\varepsilon$ )..  
 (1 +  $\varepsilon$ ) \* real (card (proj (set  $S$ ) (sols  $F$ )))}}}

**unfolding** *measure-map-pmf compute-t-find-t*

by (auto intro!: *measure-pmf.finite-measure-mono simp add: certcheck'[OF  $S$ ]*)

**also have** ...  $\leq \delta$

by (*intro approxmcL-sound'[OF assms]*)

**finally show** *?thesis* by auto

qed

**lemma** *certcheck-promise-complete*:

**assumes**  $\delta$ :  $\delta > 0$   $\delta < 1$

**assumes**  $\varepsilon$ :  $\varepsilon > 0$

**assumes**  $S$ : *distinct*  $S$

**assumes**  $r$ :  $\bigwedge r$ .

$r \in$  set-pmf (random-seed-xors (find-t  $\delta$ ) (length  $S$ ))  $\implies$   
 $\neg$ isl (certcheck  $F S \varepsilon \delta$  ( $f r$ )  $r$ )

**shows**

*measure-pmf.prov*  
 (map-pmf ( $\lambda r$ . certcheck  $F S \varepsilon \delta$  ( $f r$ )  $r$ )  
 (random-seed-xors (find-t  $\delta$ ) (length  $S$ )))

```

      {c. real (projr c) ∈
        {real (card (proj (set S) (sols F))) / (1 + ε)..
          (1 + ε) * real (card (proj (set S) (sols F)))}} ≥ 1 - δ
proof –
  have
    measure-pmf.prov
    (map-pmf (λr. certcheck F S ε δ (f r) r)
      (random-seed-xors (find-t δ) (length S)))
    {c. real (projr c) ∈
      {real (card (proj (set S) (sols F))) / (1 + ε)..
        (1 + ε) * real (card (proj (set S) (sols F)))}} =
    measure-pmf.prov
    (map-pmf (approxmc-mapL F S ε δ (find-t δ))
      (random-seed-xors (appmc.compute-t δ (find-t δ)) (length S)))
    {c. real c ∈
      {real (card (proj (set S) (sols F))) / (1 + ε)..
        (1 + ε) * real (card (proj (set S) (sols F)))}}
    unfolding measure-map-pmf compute-t-find-t
    by (auto intro!: measure-pmf.measure-pmf-eq simp add: certcheck'[OF
S] r)
    also have ... ≥ 1 - δ
    by (intro approxmcL-sound[OF assms(1-4)])
    finally show ?thesis by auto
qed

```

**end**

```

lemma certcheck-code[code]:
  certcheck F S ε δ (m0,ms) xorsLs = (
    if δ > 0 ∧ δ < 1 ∧ ε > 0 ∧ distinct S then
      (let ε = appmc.mk-eps ε in
        let thresh = appmc.compute-thresh ε in
        case BSAT F S thresh m0 of Inl err ⇒ Inl err
        | Inr Y ⇒
          if Y < thresh then Inr Y
          else
            let t = find-t δ in
              calc-median F S thresh t ms xorsLs)
    else Code.abort (STR "invalid inputs")
    (λ-. certcheck F S ε δ (m0,ms) xorsLs))
by auto

```

**end**

**end**

**end**

## 8 ApproxMC certification for CNF-XOR

This concretely instantiates the locales with a syntax and semantics for CNF-XOR, giving us a certificate checker for approximate counting in this theory.

```
theory CertCheck-CNF-XOR imports
  ApproxMCAnalysis
  CertCheck
  HOL.String HOL-Library.Code-Target-Numeral
  Show.Show-Real
begin
```

This follows CryptoMiniSAT's CNF-XOR formula syntax. A clause is a list of literals (one of which must be satisfied). An XOR constraint has the form  $l_1 + l_2 + \dots + l_n = 1$  where addition is taken over  $F_2$ . Syntactically, they are specified by the list of LHS literals. Variables are natural numbers (in practice, variable 0 is never used)

```
datatype lit = Pos nat | Neg nat
type-synonym clause = lit list
type-synonym cmsxor = lit list
type-synonym fml = clause list  $\times$  cmsxor list
```

```
type-synonym assignment = nat  $\Rightarrow$  bool
```

```
definition sat-lit :: assignment  $\Rightarrow$  lit  $\Rightarrow$  bool where
  sat-lit w l = (case l of Pos x  $\Rightarrow$  w x | Neg x  $\Rightarrow$   $\neg$ w x)
```

```
definition sat-clause :: assignment  $\Rightarrow$  clause  $\Rightarrow$  bool where
  sat-clause w C = ( $\exists$  l  $\in$  set C. sat-lit w l)
```

```
definition sat-cmsxor :: assignment  $\Rightarrow$  cmsxor  $\Rightarrow$  bool where
  sat-cmsxor w C = odd ((sum-list (map (of-bool  $\circ$  (sat-lit w)) C))::nat)
```

```
definition sat-fml :: assignment  $\Rightarrow$  fml  $\Rightarrow$  bool
where
  sat-fml w f = (
    ( $\forall$  C  $\in$  set (fst f). sat-clause w C)  $\wedge$ 
    ( $\forall$  C  $\in$  set (snd f). sat-cmsxor w C))
```

```
definition sols :: fml  $\Rightarrow$  assignment set
where sols f = {w. sat-fml w f}
```

```
lemma sat-fml-cons[simp]:
shows
  sat-fml w (FC, x # FX)  $\longleftrightarrow$ 
  sat-fml w (FC,FX)  $\wedge$  sat-cmsxor w x
```

$sat\text{-}fml\ w\ (c\ \# \ FC, \ FX) \longleftrightarrow$   
 $sat\text{-}fml\ w\ (FC, FX) \wedge sat\text{-}clause\ w\ c$   
**unfolding** *sat-fml-def by auto*

**fun** *enc-xor* ::  $nat\ xor \Rightarrow fml \Rightarrow fml$   
**where**  
 $enc\text{-}xor\ (x, b)\ (FC, FX) =$   
 $if\ b\ then\ (FC, map\ Pos\ x\ \# \ FX)$   
 $else$   
 $case\ x\ of$   
 $\ [] \Rightarrow (FC, FX)$   
 $| (v\ \# \ vs) \Rightarrow (FC, (Neg\ v\ \# \ map\ Pos\ vs)\ \# \ FX))$

**lemma** *sols-enc-xor*:  
**shows**  $sols\ (enc\text{-}xor\ (x, b)\ (FC, FX)) =$   
 $sols\ (FC, FX) \cap \{\omega.\ satisfies\ xorL\ (x, b)\ \omega\}$   
**unfolding** *sols-def*  
**by** (*cases x; auto simp add: satisfies-xorL-def sat-cmsxor-def o-def sat-lit-def list.case-eq-if*)

**definition** *check-sol* ::  $fml \Rightarrow (nat \Rightarrow bool) \Rightarrow bool$   
**where**  $check\text{-}sol\ fml\ w =$   
 $list\ all\ (list\ ex\ (sat\text{-}lit\ w))\ (fst\ fml) \wedge$   
 $list\ all\ (sat\text{-}cmsxor\ w)\ (snd\ fml)$

**definition** *ban-sol* ::  $(nat \times bool)\ list \Rightarrow fml \Rightarrow fml$   
**where**  $ban\text{-}sol\ vs\ fml =$   
 $((map\ (\lambda(v, b).\ if\ b\ then\ Neg\ v\ else\ Pos\ v)\ vs)\ \# \ fst\ fml, snd\ fml)$

**lemma** *check-sol-sol*:  
**shows**  $w \in sols\ F \longleftrightarrow$   
 $check\text{-}sol\ F\ w$   
**unfolding** *check-sol-def sols-def sat-fml-def*  
**apply** *clarsimp*  
**by** (*metis Ball-set-list-all Bex-set-list-ex sat-clause-def*)

**lemma** *ban-sat-clause*:  
**shows**  $sat\text{-}clause\ w\ (map\ (\lambda(v, b).\ if\ b\ then\ Neg\ v\ else\ Pos\ v)\ vs)$   
 $\longleftrightarrow$   
 $map\ w\ (map\ fst\ vs) \neq map\ snd\ vs$   
**unfolding** *sat-clause-def*  
**by** (*force simp add: sat-lit-def split: if-splits*)

**lemma** *sols-ban-sol*:  
**shows**  $sols\ (ban\text{-}sol\ vs\ F) =$   
 $sols\ F \cap$   
 $\{\omega.\ map\ \omega\ (map\ fst\ vs) \neq map\ snd\ vs\}$

**unfolding** *ban-sol-def sols-def*  
**by** (*auto simp add: ban-sat-clause*)

**global-interpretation** *CertCheck-CNF-XOR* :

*CertCheck sols enc-xor check-sol ban-sol*

**defines**

*random-seed-xors* = *CertCheck-CNF-XOR.random-seed-xors* **and**

*fix-t* = *CertCheck-CNF-XOR.appmc.fix-t* **and**

*find-t* = *CertCheck-CNF-XOR.find-t* **and**

*BSAT* = *CertCheck-CNF-XOR.BSAT* **and**

*check-BSAT-sols* = *CertCheck-CNF-XOR.check-BSAT-sols* **and**

*size-xorL-cert* = *CertCheck-CNF-XOR.size-xorL-cert* **and**

*approxcore-xorsL* = *CertCheck-CNF-XOR.approxcore-xorsL* **and**

*fold-approxcore-xorsL-cert* = *CertCheck-CNF-XOR.fold-approxcore-xorsL-cert*

**and**

*approxcore-xorsL-cert* = *CertCheck-CNF-XOR.approxcore-xorsL-cert*

**and**

*calc-median* = *CertCheck-CNF-XOR.calc-median* **and**

*certcheck* = *CertCheck-CNF-XOR.certcheck*

**apply** *unfold-locales*

**subgoal by** (*metis sols-enc-xor surj-pair*)

**subgoal by** (*metis sols-ban-sol*)

**by** (*metis check-sol-sol*)

## 8.1 Blasting XOR constraints to CNF

This formalizes the usual linear conversion from CNF-XOR into CNF. It is not necessary to use this conversion for solvers that support CNF-XOR formulas natively.

**definition** *negate-lit* :: *lit*  $\Rightarrow$  *lit*

**where** *negate-lit* *l* = (*case l of Pos x*  $\Rightarrow$  *Neg x* | *Neg x*  $\Rightarrow$  *Pos x*)

**fun** *xor-clauses* :: *cmsxor*  $\Rightarrow$  *bool*  $\Rightarrow$  *clause list*

**where**

*xor-clauses* [] *b* = (*if b then* [] *else* [])

| *xor-clauses* (*x#xs*) *b* =

(*let p-x* = *xor-clauses xs b* *in*

*let n-x* = *xor-clauses xs* ( $\neg$ *b*) *in*

*map* ( $\lambda c. x \# c$ ) *p-x* @ *map* ( $\lambda c. \text{negate-lit } x \# c$ ) *n-x*)

**lemma** *sat-cmsxor-nil[simp]*:

**shows**  $\neg$  (*sat-cmsxor w* [])

**unfolding** *sat-cmsxor-def*

**by** *auto*

**lemma** *sat-cmsxor-cons*:

**shows** *sat-cmsxor w* (*x # xs*) =

(if sat-lit w x then  $\neg$  (sat-cmsxor w xs) else sat-cmsxor w xs)  
**unfolding** sat-cmsxor-def  
**by** auto

**lemma** sat-cmsxor-append:  
**shows** sat-cmsxor w (xs @ ys) =  
 (if sat-cmsxor w xs then  $\neg$  (sat-cmsxor w ys) else sat-cmsxor w ys)  
**proof** (induction xs)  
**case** Nil  
**then show** ?case  
**by** (auto simp add: sat-cmsxor-def)  
**next**  
**case** (Cons x xs)  
**then show** ?case  
**by** (auto simp add: sat-cmsxor-cons)  
**qed**

**definition** sat-clauses:: assignment  $\Rightarrow$  clause list  $\Rightarrow$  bool  
**where** sat-clauses w cs = ( $\forall c \in$  set cs. sat-clause w c)

**lemma** sat-clauses-append:  
**shows** sat-clauses w (xs @ ys) =  
 (sat-clauses w xs  $\wedge$  sat-clauses w ys)  
**unfolding** sat-clauses-def **by** auto

**lemma** sat-clauses-map:  
**shows** sat-clauses w (map ((#) x) cs) =  
 (sat-lit w x  $\vee$  sat-clauses w cs)  
**unfolding** sat-clauses-def sat-clause-def **by** auto

**lemma** sat-lit-negate-lit[simp]:  
 sat-lit w (negate-lit l) = ( $\neg$ sat-lit w l)  
**apply** (cases l)  
**by** (auto simp add: negate-lit-def sat-lit-def)

**lemma** sols-xor-clauses:  
**shows**  
 sat-clauses w (xor-clauses xs b)  $\longleftrightarrow$   
 (sat-cmsxor w xs = b)  
**proof** (induction xs arbitrary: b)  
**case** Nil  
**then show** ?case  
**by** (auto simp add: sat-cmsxor-def sat-clauses-def sat-clause-def)  
**next**  
**case** (Cons x xs b)  
**have** \*: (sat-cmsxor w (x # xs) = b) =  
 (if sat-lit w x  
 then (sat-cmsxor w xs = ( $\neg$ b))  
 else (sat-cmsxor w xs = b)) **unfolding** sat-cmsxor-cons

**by** *auto*  
**have** *sat-clauses*  $w$  (*xor-clauses* ( $x \# xs$ )  $b$ )  $\longleftrightarrow$   
 $((\text{sat-lit } w \ x \vee \text{sat-clauses } w \ (\text{xor-clauses } xs \ b)) \wedge$   
 $(\neg(\text{sat-lit } w \ x) \vee \text{sat-clauses } w \ (\text{xor-clauses } xs \ (\neg b))))$   
**unfolding** *xor-clauses.simps* *Let-def* *sat-clauses-append* *sat-clauses-map*  
**by** *auto*  
**moreover have** ... = (  
 $(\text{sat-lit } w \ x \vee (\text{sat-cmsxor } w \ xs = b)) \wedge$   
 $(\neg(\text{sat-lit } w \ x) \vee (\text{sat-cmsxor } w \ xs = (\neg b))))$   
**using** *Cons.IH* **by** *auto*  
**moreover have** ... = (*sat-cmsxor*  $w$  ( $x \# xs$ ) =  $b$ )  
**unfolding** \* **by** *auto*  
**ultimately show** ?*case*  
**by** *auto*  
**qed**

**definition** *var-lit* :: *lit*  $\Rightarrow$  *nat*  
**where** *var-lit*  $l = (\text{case } l \text{ of } \text{Pos } x \Rightarrow x \mid \text{Neg } x \Rightarrow x)$

**definition** *var-lits* :: *lit list*  $\Rightarrow$  *nat*  
**where** *var-lits*  $ls = \text{fold } \text{max} \ (\text{map } \text{var-lit } ls) \ 0$

**lemma** *sat-lit-same*:  
**assumes**  $\bigwedge x. x \leq \text{var-lit } l \implies w \ x = w' \ x$   
**shows** *sat-lit*  $w \ l = \text{sat-lit } w' \ l$   
**using** *assms*  
**apply** (*cases*  $l$ )  
**by** (*auto simp add: sat-lit-def var-lit-def*)

**lemma** *var-lits-eq*:  
 $\text{var-lits } ls = \text{Max} \ (\text{set } (0 \# \text{map } \text{var-lit } ls))$   
**unfolding** *var-lits-def* *Max.set-eq-fold*  
**by** *auto*

**lemma** *sat-lits-same*:  
**assumes**  $\bigwedge x. x \leq \text{var-lits } c \implies w \ x = w' \ x$   
**shows** *sat-clause*  $w \ c = \text{sat-clause } w' \ c$   
**using** *assms*  
**unfolding** *sat-clause-def* *var-lits-eq* *sat-lit-same*  
**by** (*smt (verit) List.finite-set Max-ge dual-order.trans image-subset-iff*  
*list.set-map sat-lit-same set-subset-Cons*)

**lemma** *le-var-lits-in*:  
**assumes**  $y \in \text{set } ys \ v \leq \text{var-lit } y$   
**shows**  $v \leq \text{var-lits } ys$   
**using** *assms* **unfolding** *var-lits-eq*  
**by** (*metis List.finite-set Max-ge dual-order.trans imageI insertCI*)



*list.set(2) list.set-map*)

**lemma** *sat-cmsxor-same*:

**assumes**  $\bigwedge x. x \leq \text{var-lits } xs \implies w x = w' x$

**shows**  $\text{sat-cmsxor } w \text{ } xs = \text{sat-cmsxor } w' \text{ } xs$

**using** *assms*

**proof** (*induction xs*)

**case** *Nil*

**then show** *?case*

**unfolding** *sat-cmsxor-def*

**by** *auto*

**next**

**case** *ih:(Cons x xs)*

**have** *1: sat-lit w x = sat-lit w' x*

**apply** (*intro sat-lit-same*)

**using** *ih(2)*

**by** (*metis le-var-lits-in list.set-intros(1)*)

**have** *2: sat-cmsxor w xs = sat-cmsxor w' xs*

**apply** (*intro ih(1)*)

**using** *ih(2)*

**by** (*smt (verit) List.finite-set Max-ge-iff empty-not-insert insert-iff*

*list.set(2) list.simps(9) var-lits-eq*)

**show** *?case*

**unfolding** *sat-cmsxor-cons*

**using** *1 2 by presburger*

**qed**

**lemma** *sat-cmsxor-split*:

**assumes**  $u: \text{var-lits } xs < u \text{ var-lits } ys < u$

**assumes**  $w': w' = (\lambda x. \text{if } x = u \text{ then } \neg \text{sat-cmsxor } w \text{ } xs \text{ else } w \text{ } x)$

**shows**

$(\text{sat-cmsxor } w \text{ } (xs @ ys) =$   
 $(\text{sat-cmsxor } w' \text{ } (\text{Pos } u \# xs) \wedge$   
 $\text{sat-cmsxor } w' \text{ } (\text{Neg } u \# ys)))$

**proof** –

**have** *xs: sat-cmsxor w' xs = sat-cmsxor w xs*

**apply** (*intro sat-cmsxor-same*)

**using** *u unfolding w' by auto*

**have** *ys: sat-cmsxor w' ys = sat-cmsxor w ys*

**apply** (*intro sat-cmsxor-same*)

**using** *u unfolding w' by auto*

**show** *?thesis*

**unfolding** *sat-cmsxor-append sat-cmsxor-cons xs ys*

**unfolding** *w' by (auto simp add: sat-lit-def)*

**qed**

```

fun split-xor :: nat  $\Rightarrow$  cmsxor  $\Rightarrow$  cmsxor list  $\times$  nat  $\Rightarrow$  cmsxor list  $\times$ 
nat
  where split-xor k xs (acc,u) = (
    if length xs  $\leq$  k + 3 then (xs # acc, u)
    else (
      let xs1 = take (k + 2) xs in
      let xs2 = drop (k + 2) xs in
      split-xor k (Neg u # xs2) ((Pos u # xs1) # acc, u+1)
    )
  )

```

```

declare split-xor.simps[simp del]

```

```

lemma split-xor-bound:

```

```

  assumes split-xor k xs (acc,u) = (acc',u')
  shows u  $\leq$  u'
  using assms
proof (induction length xs arbitrary: xs acc u acc' u' rule: less-induct)
  case less
  have length xs  $\leq$  k + 3  $\vee$   $\neg$  (length xs  $\leq$  k + 3) by auto
  moreover {
    assume length xs  $\leq$  k + 3
    then have u  $\leq$  u'
      using less(2) split-xor.simps by auto
  }
  moreover {
    assume s:  $\neg$  (length xs  $\leq$  k + 3)
    then have l: length (Neg u # drop (Suc (Suc k)) xs) < length xs
      by auto
    have Suc u  $\leq$  u'
      using s less(2)
      unfolding split-xor.simps[of k xs]
      using less(1)[OF l] by auto
    then have u  $\leq$  u' by auto
  }
  ultimately show ?case by auto
qed

```

```

lemma var-lits-append:

```

```

  shows var-lits xs  $\leq$  var-lits (xs @ ys)
    var-lits ys  $\leq$  var-lits (xs @ ys)
  unfolding var-lits-eq
  by auto

```

```

lemma fold-max-eq:

```

```

  assumes i  $\leq$  u
  shows fold max ls u = max u (fold max ls (i::nat))
  using assms
  apply (induction ls arbitrary: i)

```

**subgoal by** *clarsimp*  
**apply** *clarsimp*  
**by** (*smt* (*verit*) *List.finite-set Max.set-eq-fold Max-ge-iff empty-iff insert-iff list.set(2) max.orderI max-def pre-arith-simps(3)*)

**lemma** *split-xor-sound*:

**assumes** *sat-cmsxor w xs*  $\bigwedge x. x \in \text{set } acc \implies \text{sat-cmsxor } w \ x$

**assumes** *u: var-lits xs < u*  $\bigwedge x. x \in \text{set } acc \implies \text{var-lits } x < u$

**assumes** *split-xor k xs (acc,u) = (acc',u')*

**obtains** *w' where*

$\bigwedge x. x < u \implies w \ x = w' \ x$

$\bigwedge x. x \in \text{set } acc' \implies \text{sat-cmsxor } w' \ x$

$\bigwedge x. x \in \text{set } acc' \implies \text{var-lits } x < u'$

**using** *assms*

**proof** (*induction length xs arbitrary: w xs acc u acc' u' thesis rule: less-induct*)

**case** *less*

**have** *length xs ≤ k + 3*  $\vee \neg (\text{length } xs \leq k + 3)$  **by** *auto*

**moreover** {

**assume** *length xs ≤ k + 3*

**then have** *\*: acc' = xs # acc*

**using** *less(7) split-xor.simps*

**by** *auto*

**then have**

$\bigwedge x. x < u \implies w \ x = w \ x$

$\bigwedge x. x \in \text{set } acc' \implies \text{sat-cmsxor } w \ x$

$\bigwedge x. x \in \text{set } acc' \implies \text{var-lits } x < u'$

**subgoal by** *meson*

**subgoal using** *less*

**by** (*metis \* set-ConsD*)

**using** *less*

**by** (*metis \* order-trans-rules(22) set-ConsD split-xor-bound*)

}

**moreover** {

**assume** *s: ¬ (length xs ≤ k + 3)*

**define** *xs1 where xs1:xs1 = take (k + 2) xs*

**define** *xs2 where xs2:xs2 = drop (k + 2) xs*

**have** *sp: split-xor k (Neg u # xs2) ((Pos u # xs1) # acc, u+1) = (acc',u')*

**by** (*metis less(7) s split-xor.simps xs1 xs2*)

**have** *l: length (Neg u # xs2) < length xs*

**using** *s xs2* **by** *auto*

**have** *xs: xs = xs1 @ xs2* **by** (*simp add: xs1 xs2*)

**define** *w' where w': w' =*

( $\lambda x. \text{if } x = u \text{ then } \neg \text{sat-cmsxor } w \ xs1 \text{ else } w \ x$ )

```

have vl: var-lits xs1 < u var-lits xs2 < u
apply (metis less(5) order-trans-rules(21) var-lits-append(1) xs)
by (metis less(5) order-trans-rules(21) var-lits-append(2) xs)

from sat-cmsxor-split[OF this w']
have satws1: sat-cmsxor w' (Pos u # xs1)
and satws2: sat-cmsxor w' (Neg u # xs2)
using less(3) xs by auto

have satacc:  $\bigwedge x. x \in \text{set } ((\text{Pos } u \# \text{xs1}) \# \text{acc}) \implies$ 
  sat-cmsxor w' x
by (metis less(4) less(6) not-less sat-cmsxor-same satws1 set-ConsD
w')

have v1: var-lits (Neg u # xs2) < u + 1
unfolding var-lits-def
apply (simp add: var-lit-def)
using vl var-lits-def fold-max-eq
by (metis le-add2 le-add-same-cancel2 less-Suc-eq max.absorb3)

have fold max (map var-lit xs1) u < Suc u
using vl var-lits-def
by (smt (verit, best) List.finite-set Max.set-eq-fold Max-ge-iff
dual-order.trans empty-iff insert-iff lessI less-or-eq-imp-le list.set(2)
not-less)
moreover have  $\bigwedge x. x \in \text{set } \text{acc} \implies$ 
  fold max (map var-lit x) 0 < Suc u
by (metis less(6) less-SucI var-lits-def)
ultimately have v2:  $\bigwedge x. x \in \text{set } ((\text{Pos } u \# \text{xs1}) \# \text{acc}) \implies$ 
var-lits x < u + 1
unfolding var-lits-def
by (auto simp add: var-lit-def)

obtain w'' where
w'':  $\bigwedge x. x < u + 1 \implies w' x = w'' x$ 
 $\bigwedge x. x \in \text{set } \text{acc}' \implies \text{sat-cmsxor } w'' x$ 
 $\bigwedge x. x \in \text{set } \text{acc}' \implies \text{var-lits } x < u'$ 
using less(1)[OF l - satws2 satacc v1 v2 sp]
by auto

have  $\bigwedge x. x < u \implies w x = w'' x$ 
using w''(1)
unfolding w'
using less-Suc-eq by fastforce

then have  $\exists w''. ($ 
   $(\forall x. x < u \longrightarrow w x = w'' x) \wedge$ 
   $(\forall x. x \in \text{set } \text{acc}' \longrightarrow \text{sat-cmsxor } w'' x) \wedge$ 
   $(\forall x. x \in \text{set } \text{acc}' \longrightarrow \text{var-lits } x < u')$ 

```

```

    using w''(2-3) by auto
  }
  ultimately show ?case
    using less(2)
    by (metis (mono-tags, lifting))
qed

```

**definition** *split-xors* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{cmsxor list} \Rightarrow \text{cmsxor list}$   
**where** *split-xors*  $k\ u\ xs = \text{fst} (\text{fold} (\text{split-xor } k) xs ([], u))$

**lemma** *split-xors-sound*:

```

  assumes  $\bigwedge x. x \in \text{set } xs \Longrightarrow \text{sat-cmsxor } w\ x$ 
     $\bigwedge x. x \in \text{set } acc \Longrightarrow \text{sat-cmsxor } w\ x$ 
  assumes  $u: \bigwedge x. x \in \text{set } xs \Longrightarrow \text{var-lits } x < u$ 
     $\bigwedge x. x \in \text{set } acc \Longrightarrow \text{var-lits } x < u$ 
  assumes  $\text{fold} (\text{split-xor } k) xs (acc, u) = (acc', u')$ 
  obtains  $w'$  where
     $\bigwedge x. x < u \Longrightarrow w\ x = w'\ x$ 
     $\bigwedge x. x \in \text{set } acc' \Longrightarrow \text{sat-cmsxor } w'\ x$ 
     $\bigwedge x. x \in \text{set } acc' \Longrightarrow \text{var-lits } x < u'$ 
  using assms
proof (induction xs arbitrary: w acc u acc' u')
  case Nil
  then show ?case
    by auto
next
  case  $ih:(\text{Cons } x\ xs)$ 
  obtain  $acc''\ u''$  where  $x: \text{split-xor } k\ x\ (acc, u) = (acc'', u'')$ 
    by fastforce
  from split-xor-sound[OF - - - this]
  obtain  $w''$  where
     $w'': \bigwedge x. x < u \Longrightarrow w\ x = w''\ x$ 
     $\bigwedge x. x \in \text{set } acc'' \Longrightarrow \text{sat-cmsxor } w''\ x$ 
     $\bigwedge x. x \in \text{set } acc'' \Longrightarrow \text{var-lits } x < u''$ 
  by (smt (verit, del-insts) ih(4) ih(5) ih(6) ih.prem(2) list.set-intros(1))
  have  $rw: \text{fold} (\text{split-xor } k) xs (acc'', u'') = (acc', u')$ 
    using  $ih(7)\ x$  by auto
  have 1:  $\bigwedge x. x \in \text{set } xs \Longrightarrow \text{sat-cmsxor } w''\ x$ 
    by (metis basic-trans-rules(22) ih(3) ih(5) list.set-intros(2) not-less
sat-cmsxor-same w''(1))
  have 2:  $\bigwedge x. x \in \text{set } xs \Longrightarrow \text{var-lits } x < u''$ 
    by (meson ih(5) list.set-intros(2) order-trans-rules(22) split-xor-bound
x)
  show ?case
    using ?case
      using  $ih(1)$ [OF - 1  $w''(2)$  2  $w''(3)$   $rw$ ]
      by (smt (verit, best) basic-trans-rules(22) ih(2) split-xor-bound
w''(1) x)
qed

```

**definition** *var-fml* :: *fml*  $\Rightarrow$  *nat*  
**where** *var-fml* *f* =  
 $\max$  ( $\text{fold } \max$  ( $\text{map } \text{var-lits}$  ( $\text{fst } f$ )) 0)  
( $\text{fold } \max$  ( $\text{map } \text{var-lits}$  ( $\text{snd } f$ )) 0)

**lemma** *var-fml-eq*:  
 $\text{var-fml } f =$   
 $\max$  ( $\text{Max}$  ( $\text{set}$  (0 #  $\text{map } \text{var-lits}$  ( $\text{fst } f$ ))))  
( $\text{Max}$  ( $\text{set}$  (0 #  $\text{map } \text{var-lits}$  ( $\text{snd } f$ ))))  
**unfolding** *var-fml-def* *Max.set-eq-fold*  
**by** *auto*

**definition** *split-fml* :: *nat*  $\Rightarrow$  *fml*  $\Rightarrow$  *fml*  
**where** *split-fml* *k* *f* = (  
 $\text{let } u = \text{var-fml } f + 1$   $\text{in}$   
( $\text{fst } f$ , ( $\text{split-xors } k$   $u$  ( $\text{snd } f$ )))  
)

**lemma** *var-lits-var-fml*:  
**shows**  $\bigwedge x. x \in \text{set } (\text{snd } F) \Longrightarrow \text{var-lits } x \leq \text{var-fml } F$   
 $\bigwedge x. x \in \text{set } (\text{fst } F) \Longrightarrow \text{var-lits } x \leq \text{var-fml } F$   
**unfolding** *var-fml-eq*  
**subgoal**  
**apply** *clarsimp*  
**by** (*meson* *List.finite-set* *Max-ge* *finite-imageI* *finite-insert* *image-subset-iff* *max.coboundedI2* *subset-insertI*)  
**apply** *clarsimp*  
**by** (*meson* *List.finite-set* *Max-ge* *finite-imageI* *finite-insert* *image-subset-iff* *max.coboundedI1* *subset-insertI*)

**lemma** *split-fml-satisfies*:  
**assumes** *sat-fml*  $w$   $F$   
**obtains**  $w'$  **where** *sat-fml*  $w'$  ( $\text{split-fml } k$   $F$ )  
**proof** –  
**obtain**  $\text{acc}'$   $u'$  **where**  
 $*$ :  $\text{fold } (\text{split-xor } k) (\text{snd } F) ([], \text{var-fml } F + 1) = (\text{acc}', u')$   
**by** *fastforce*  
**have** 1:  $\bigwedge x. x \in \text{set } (\text{snd } F) \Longrightarrow \text{sat-cmsxor } w$   $x$   
**using** *assms* **unfolding** *sat-fml-def* **by** *auto*  
**have** 2:  $\bigwedge x. x \in \text{set } (\text{snd } F) \Longrightarrow \text{var-lits } x < \text{var-fml } F + 1$   
**using** *var-lits-var-fml*  
**by** (*meson* *less-add-one* *order-le-less-trans*)  
**from** *split-xors-sound*[*OF* 1 - - 2  $*$ ]  
**obtain**  $w'$  **where**  
 $w'$ :  $\bigwedge x. x < \text{var-fml } F + 1 \Longrightarrow w$   $x = w'$   $x$   
 $\bigwedge x. x \in \text{set } \text{acc}' \Longrightarrow \text{sat-cmsxor } w'$   $x$   
 $\bigwedge x. x \in \text{set } \text{acc}' \Longrightarrow \text{var-lits } x < u'$   
**using** 2 **by** *auto*

**have**  $\bigwedge x. x \in \text{set } (\text{fst } F) \implies \text{sat-clause } w' x$   
**by** (*metis* *assms* *less-add-one* *order-trans-rules*(21) *sat-fml-def* *sat-lits-same* *var-lits-var-fml*(2) *w'(1)*)

**then have** *sat-fml* *w'* (*split-fml* *k* *F*)  
**unfolding** *split-fml-def* *Let-def* *split-xors-def* \*  
**by** (*auto simp add: sat-fml-def w'(2)*)  
**thus** *?thesis*  
**using** *that* **by** *auto*  
**qed**

**lemma** *split-fml-sols*:  
**assumes** *sols* (*split-fml* *k* *F*) = {}  
**shows** *sols* *F* = {}  
**using** *assms*  
**using** *split-fml-satisfies* **unfolding** *sols-def*  
**by** (*metis* *Collect-empty-eq*)

**definition** *blast-xors* :: *cmsxor list*  $\Rightarrow$  *clause list*  
**where** *blast-xors* *xors* = *concat* (*map* ( $\lambda x. \text{xor-clauses } x \text{ True}$ ) *xors*)

**definition** *blast-fml* :: *fml*  $\Rightarrow$  *clause list*  
**where** *blast-fml* *f* =  
*fst* *f* @ *blast-xors* (*snd* *f*)

**lemma** *sat-clauses-concat*:  
*sat-clauses* *w* (*concat* *xs*)  $\longleftrightarrow$   
 $(\forall x \in \text{set } xs. \text{sat-clauses } w x)$   
**unfolding** *sat-clauses-def*  
**by** *auto*

**lemma** *blast-xors-sound*:  
**assumes**  $(\bigwedge x. x \in \text{set } xors \implies \text{sat-cmsxor } w x)$   
**shows** *sat-clauses* *w* (*blast-xors* *xors*)  
**unfolding** *blast-xors-def* *sat-clauses-concat*  
**by** (*auto simp add: assms sols-xor-clauses*)

**lemma** *blast-fml-sound*:  
**assumes** *sat-fml* *w* *F*  
**shows** *sat-fml* *w* (*blast-fml* *F*, [])  
**unfolding** *blast-fml-def* *sat-fml-def*  
**apply** *clarsimp*  
**using** *assms* *blast-xors-sound* *sat-clauses-def* *sat-fml-def* **by** *blast*

**definition** *blast-split-fml* :: *fml*  $\Rightarrow$  *clause list*  
**where** *blast-split-fml* *f* = *blast-fml* (*split-fml* 1 *f*)

**lemma** *blast-split-fml-sols*:  
**assumes** *sols* (*blast-split-fml* *F*, []) = {}

**shows**  $\text{sols } F = \{\}$   
**by** (*metis Collect-empty-eq assms blast-fml-sound blast-split-fml-def*  
*sols-def split-fml-sols*)

**definition** *certcheck-blast::*

(*clause list*  $\Rightarrow$  *bool*)  $\Rightarrow$   
*fml*  $\Rightarrow$  *nat list*  $\Rightarrow$   
*real*  $\Rightarrow$  *real*  $\Rightarrow$   
((*nat*  $\Rightarrow$  *bool*) *list*  $\times$   
(*nat*  $\Rightarrow$  (*nat*  $\times$  (*nat*  $\Rightarrow$  *bool*) *list*  $\times$  (*nat*  $\Rightarrow$  *bool*) *list*)))  $\Rightarrow$   
(*nat*  $\Rightarrow$  *nat*  $\Rightarrow$  (*bool list*  $\times$  *bool*))  $\Rightarrow$   
*String.literal* + *nat*  
**where** *certcheck-blast check-unsat*  $F S \varepsilon \delta m0ms =$   
*certcheck (check-unsat  $\circ$  blast-split-fml)*  $F S \varepsilon \delta m0ms$

**corollary** *certcheck-blast-sound:*

**assumes**  $\bigwedge F. \text{check-unsat } F \Longrightarrow \text{sols } (F, []) = \{\}$   
**assumes**  $0 < \delta \delta < 1$   
**assumes**  $0 < \varepsilon$   
**assumes** *distinct*  $S$   
**shows**  
*measure-pmf.prov*  
(*map-pmf* ( $\lambda r. \text{certcheck-blast check-unsat } F S \varepsilon \delta (f r) r$ )  
(*random-seed-xors (find-t*  $\delta)$  (*length*  $S$ )))  
 $\{c. \neg \text{isl } c \wedge$   
*real (projr*  $c) \notin$   
 $\{\text{real (card (proj (set } S) (\text{sols } F)))} / (1 + \varepsilon)..$   
 $(1 + \varepsilon) * \text{real (card (proj (set } S) (\text{sols } F)))}\} \leq \delta$   
**unfolding** *certcheck-blast-def*  
**apply** (*intro CertCheck-CNF-XOR.certcheck-sound[OF - assms(2-5)]*)  
**using** *assms(1) unfolding Let-def*  
**using** *blast-split-fml-sols by auto*

**corollary** *certcheck-blast-promise-complete:*

**assumes**  $\bigwedge F. \text{check-unsat } F \Longrightarrow \text{sols } (F, []) = \{\}$   
**assumes**  $0 < \delta \delta < 1$   
**assumes**  $0 < \varepsilon$   
**assumes** *distinct*  $S$   
**assumes**  $r: \bigwedge r.$   
 $r \in \text{set-pmf (random-seed-xors (find-t } \delta) (\text{length } S)) \Longrightarrow$   
 $\neg \text{isl (certcheck-blast check-unsat } F S \varepsilon \delta (f r) r)$   
**shows**  
*measure-pmf.prov*  
(*map-pmf* ( $\lambda r. \text{certcheck-blast check-unsat } F S \varepsilon \delta (f r) r$ )  
(*random-seed-xors (find-t*  $\delta)$  (*length*  $S$ )))  
 $\{c. \text{real (projr } c) \in$   
 $\{\text{real (card (proj (set } S) (\text{sols } F)))} / (1 + \varepsilon)..$   
 $(1 + \varepsilon) * \text{real (card (proj (set } S) (\text{sols } F)))}\} \geq 1 - \delta$



```

unfolding certcheck-blast-def
apply (intro CertCheck-CNF-XOR.certcheck-promise-complete[OF -
assms(2-5)])
using assms(1) unfolding Let-def
using blast-split-fml-sols assms(6) unfolding certcheck-blast-def by
auto

```

## 8.2 Export code for a SML implementation.

```

definition real-of-int :: integer  $\Rightarrow$  real
where real-of-int n = real (nat-of-integer n)

```

```

definition real-mult :: real  $\Rightarrow$  real  $\Rightarrow$  real
where real-mult n m = n * m

```

```

definition real-div :: real  $\Rightarrow$  real  $\Rightarrow$  real
where real-div n m = n / m

```

```

definition real-plus :: real  $\Rightarrow$  real  $\Rightarrow$  real
where real-plus n m = n + m

```

```

definition real-minus :: real  $\Rightarrow$  real  $\Rightarrow$  real
where real-minus n m = n - m

```

```

declare [[code abort: fix-t]]

```

### export-code

```

length
nat-of-integer int-of-integer
integer-of-nat integer-of-int
real-of-int real-mult real-div real-plus real-minus
quotient-of

```

```

Pos Neg
CertCheck-CNF-XOR.appmc.compute-thresh
find-t certcheck
certcheck-blast
in SML

```

```

end

```

## References

- [1] S. Chakraborty, K. S. Meel, and M. Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In S. Kambhampati, editor, *IJ-CAI*, pages 3569–3576. IJCAI/AAAI Press, 2016.