Arbitrage Opportunities Correspond to Probability Inequality Identities

Matthew Doty

February 4, 2026

Abstract

We consider a fixed-odds gambling market over arbitrary logical propositions, where participants trade bets involving conjunctions, disjunctions, and negations. In this setting, we establish a three-way correspondence between the financial feasibility of trading strategies, the validity of universal probability inequalities, and the solutions to bounded Maximum Satisfiability (MaxSAT) problems.

The central result demonstrates that proving a trading strategy constitutes an arbitrage opportunity (i.e., guaranteeing a risk-free profit regardless of the outcome) is equivalent to proving a specific inequality identity holds for all probability functions, and is computationally equivalent to establishing a lower bound on a corresponding MaxSAT instance. Dually, we show that checking the coherence of a strategy (i.e., ensuring it does not guarantee a loss) also corresponds to verifying a probability identity and bounding a MaxSAT problem from above.

Contents

1 Introduction

theory *Arbitrage-Probability-Correspondence*
imports
Probability-Inequality-Completeness.Probability-Inequality-Completeness
HOL.Real
begin

1.1 Motivation

Consider a *fixed-odds gambling market* where participants trade bets on arbitrary logical propositions.

In this setting, every bet pays out exactly $*1* if the proposition is true and $*0* otherwise. Unlike traditional prediction markets like *PredictIt* or *Polymarket*, which usually limit trading to mutually exclusive outcomes, we assume a market that allows bets on any combination of logical operators: *AND* ($\sqcap$), *OR* ($\sqcup$), and *NOT* ($\sim$).

To understand the relationship between market liquidity and probability logic, imagine two events:

- *A* :: *The NASDAQ will go up 1% by Friday*
- *B* :: *The S&P500 will go up 1% by Friday*

Suppose the market order book contains the following quotes:

- *ASK* for *A* at $*0.40* (Someone is selling/offering a bet on *A*).
- *ASK* for *B* at $*0.50* (Someone is selling/offering a bet on *B*).
- *BID* for $A \sqcap B$ at $*0.30* (Someone wants to buy a bet on *A AND B*).
- *BID* for $A \sqcup B$ at $*0.70* (Someone wants to buy a bet on *A OR B*).

An arbitrageur can exploit these prices to guarantee a risk-free profit.

They act as a *market taker* for the *ASK*s (buying A and B) and as a *market maker* for the BIDs (selling A *AND* B and A *OR* B).

The initial cash flow is positive:

$Profit = (BID(A \sqcap B) + BID(A \sqcup B)) - (ASK(A) + ASK(B))$ $Profit = (\$0.30 + \$0.70) - (\$0.40 + \$0.50) = \$1.00 - \$0.90 = \$0.10$

Crucially, this profit is safe regardless of the outcome. The arbitrageur holds long positions in A and B, and short positions in $A \sqcap B$ and $A \sqcup B$.

- If both rise (True, True): The arbitrageur wins \$*2* on longs, pays \$*2* on shorts. Net: \$*0* payout.
- If only one rises (True, False): The arbitrageur wins \$*1* on longs, pays \$*1* on short (the *OR* bet). Net: \$*0* payout.
- If neither rises (False, False): The arbitrageur wins \$*0*, pay \$*0*. Net: \$*0* payout.

  The arbitrage exists because the market prices violate the probability identity:

  $Pr(A) + Pr(B) = Pr(A \sqcap B) + Pr(A \sqcup B)$

  The central result of this work generalizes this intuition:

  Every arbitrage opportunity corresponds to a probability inequality identity.

1.2 Overview of Results

The central result of this work is as follows:

Proving a strategy will always yield a profit (if completely matched) in a fixed-odds gambling market over arbitrary logical propositions corresponds to proving an inequality identity in probability logic, and also corresponds to a bounded MaxSAT problem.

Such strategies are referred to as *arbitrage strategies.*

We also consider the *dual* problem of identifying if a trading strategy will never make a profit. Strategies that will never logically yield a profit are called *incoherent.*

1.3 Prior Work

Two results that appear to be related at first glance are *The Fundamental Theorem(s) of Asset Pricing* (FTAP) [6] and the *Dutch Book Theorem* [1, 3, 4, 5]. While the connection to FTAP is purely superficial, the results are close in spirit to the Dutch Book tradition: we study when a collection of fixed-odds commitments can be combined into a strategy that is guaranteed to lose (or, dually, guaranteed to profit), and we treat such strategies as computational objects.

The Fundamental Theorems of Asset Pricing (FTAP) connect a suitable notion of *no-arbitrage* to the existence of a pricing functional (or, in stochastic settings, an equivalent martingale measure) in an idealized, frictionless market. In their classical formulations, the objects being priced are standard financial assets (e.g., securities or commodities) represented by a spot price or a price process, and the market model abstracts away from microstructure: order placement, order matching, bid/ask discreteness, and fixed-odds quoting are not part of the primitive data. By contrast, we work directly with fixed-odds markets for wagers on arbitrary logical propositions, where the microstructure of how orders compose into strategies is central, and we connect "no-arbitage" strategies to the existence of some scenario where the strategy doesn't always lose, which falls out of a certain bounded MaxSAT calculation.

The Dutch Book literature shares more of our vocabulary. Philosophical treatments emphasize *coherence* and the avoidance of a *bad book*: a collection of bets that guarantees a loss. Following Hájek's terminology [2], one may also speak of *good books*. In this development, we adopt finance-oriented language and refer to these objects as (loss-guaranteeing) *arbitrage strategies*, because they are assembled from posted odds and executed mechanically once the relevant orders are matched. We also work with possibility-style representations in the spirit of Lehman, generalized to any instance of a *classical-logic*.

Our main contribution is not a normative thesis that rational agents ought to conform their degrees of belief to probability theory. Instead, we make explicit a three-way correspondence between:

1. checking whether a bounded family of fixed-odds commitments is coherent (i.e., not loss-guaranteeing),
2. feasibility of a bounded MaxSAT instance derived from the same commitments, and
3. certain inequalities that hold for all probability functions over the same set of propositions.

Operationally, we only require the first criterion: there must exist a scenario in which the strategy does not always lose. The MaxSAT formulation supplies a concrete decision procedure, and the coNP-hardness of the resulting feasibility questions explains why coherence checking is not a task one should expect to perform reliably by hand.

We also study the *dual* problem: identifying strategies that are pure arbitrages (guaranteed nonnegative payoff with strictly positive payoff in some outcome). Such strategies are useful not merely as pathologies, but as mechanisms for creating market depth. Intuitively, they can match *BID* interest in one venue with *ASK* interest in another, improving execution for multiple participants. From a microeconomic perspective, this can increase surplus by enabling trades that would otherwise fail to clear.

2 Fixed Odds Markets

notation *Probability-Inequality-Completeness.relative-maximals* (‹$\mathcal{M}$›)

unbundle *no funcset-syntax*

2.1 Orders and Trading Strategies

In this section, we model a *fixed odds market* where each bet pays out \$*0* or \$*1*, and people make and take bets. For simplicity, we consider *BID* and *ASK* limit orders of a single unit (i.e., trades such that if they match, then they are completely cleared). In an ordinary central limit order book, such *BID* and *ASK* orders would have prices in the interval (*0*,*1*), but we do not make use of this assumption in our proofs, as it is not necessary.

record *'p bet-offer* =
  bet :: *'p*
  price :: *real*

A trading strategy is a collection of *BID* and *ASK* orders that are to be matched atomically.

Making a bet is when you *ask* a bet on the market, while *taking* a bet is when you *bid* a bet on the market.

A *market maker* is one who puts up capital and asks bets, while a *market taker* is one who bids bets.

In a trading strategy, the market participant acts as a market maker for the *ASK* orders they are willing make and as a market taker for the *BID* orders they are willing to make.

record *'p strategy* =
  asks :: (*'p bet-offer*) *list*
  bids :: (*'p bet-offer*) *list*

2.2 Possibility Functions

Possibility functions are states of affairs that determine the outcomes of bets. They were first used in Lehman's formulation of the Dutch Book

Theorem [4]. Our approach diverges from Lehman's. Lehman uses linear programming to prove his result. Our formulation is pure probability logic.

We give our definition of a possibility function as follows:

```
definition (in classical-logic) possibility :: ('a ⇒ bool) ⇒ bool where
  [simp]: possibility p ≡
              ¬ (p ⊥)
            ∧ (∀ φ. ⊢ φ ⟶ p φ)
            ∧ (∀ φ ψ . p (φ → ψ) ⟶ p φ ⟶ p ψ)
            ∧ (∀ φ . p φ ∨ p (∼ φ))
```

Our formulation of possibility functions generalizes Lehman's. Lehman restricts his definition to the language of classical propositional logic formulae. We define ours over any arbitrary classical logic satisfying the axioms of the *classical-logic* class.

```
definition (in classical-logic) possibilities :: ('a ⇒ bool) set where
  [simp]: possibilities = {p. possibility p}

lemma (in classical-logic) possibility-negation:
  assumes possibility p
  shows p (φ → ⊥) = (¬ p φ)
proof
  assume p (φ → ⊥)
  show ¬ p φ
  proof
    assume p φ
    have ⊢ φ → (φ → ⊥) → ⊥
      by (simp add: double-negation-converse)
    hence p ((φ → ⊥) → ⊥)
      using ‹p φ› ‹possibility p› by auto
    thus False using ‹p (φ → ⊥)› ‹possibility p› by auto
  qed
next
  show ¬ p φ ⟹ p (φ → ⊥)
    using ‹possibility p› negation-def by fastforce
qed

lemma (in classical-logic) possibilities-logical-closure:
  assumes possibility p
      and {x. p x} ⊩ φ
    shows p φ
proof –
  {
    fix Γ
    assume set Γ ⊆ Collect p
    hence ∀ φ. Γ :⊢ φ ⟶ p φ
    proof (induct Γ)
      case Nil
      have ∀φ. ⊢ φ ⟶ p φ
```

```
      using ‹possibility p› by auto
    then show ?case
      using list-deduction-base-theory by blast
  next
    case (Cons γ Γ)
    hence p γ
      by simp
    have ∀ φ. Γ :⊢ γ → φ ⟶ p (γ → φ)
      using Cons.hyps Cons.prems by auto
    then show ?case
      by (meson
            ‹p γ›
            ‹possibility p›
            list-deduction-theorem
            possibility-def)
  qed
 }
 thus ?thesis
   using ‹Collect p ⊩ φ› set-deduction-def by auto
qed
```

The next two lemmas establish that possibility functions are equivalent to maximally consistent sets.

```
lemma (in classical-logic) possibilities-are-MCS:
 assumes possibility p
 shows MCS {x. p x}
 using assms
 by (metis
      (mono-tags, lifting)
      formula-consistent-def
      formula-maximally-consistent-set-def-def
      maximally-consistent-set-def
      possibilities-logical-closure
      possibility-def
      mem-Collect-eq
      negation-def)

lemma (in classical-logic) MCSs-are-possibilities:
 assumes MCS s
 shows possibility (λ x. x ∈ s)
proof −
 have ⊥ ∉ s
   using ‹MCS s›
        formula-consistent-def
        formula-maximally-consistent-set-def-def
        maximally-consistent-set-def
        set-deduction-reflection
   by blast
 moreover have ∀ φ. ⊢ φ ⟶ φ ∈ s
```

using ‹*MCS s*›
formula-maximally-consistent-set-def-reflection
maximally-consistent-set-def
set-deduction-weaken
by *blast*
moreover have $\forall\ \varphi\ \psi.\ (\varphi \to \psi) \in s \longrightarrow \varphi \in s \longrightarrow \psi \in s$
using ‹*MCS s*›
formula-maximal-consistency
formula-maximally-consistent-set-def-implication
by *blast*
moreover have $\forall\ \varphi.\ \varphi \in s \lor (\varphi \to \bot) \in s$
using *assms*
formula-maximally-consistent-set-def-implication
maximally-consistent-set-def
by *blast*
ultimately show *?thesis* **by** (*simp add*: *negation-def*)
qed

2.3 Payoff Functions

Given a market strategy and a possibility function, we can define the *payoff* of that strategy if all the bet positions in that strategy were matched and settled at the particular state of affairs given by the possibility function.

Recall that in a trading strategy, we act as a market *maker* for ask positions, meaning we payout if the proposition behind the bet we are asking evaluates to *true*.

Payoff is revenue from won bets minus costs of the *BID*s for those bets, plus revenue from sold *ASK* bets minus payouts from bets lost.

definition *payoff* :: ($'p \Rightarrow bool$) $\Rightarrow$ $'p$ *strategy* $\Rightarrow$ *real* (π) **where**
[*simp*]: π *s strategy* $\equiv$
($\sum$ *i* $\leftarrow$ *bids strategy*. (*if s* (*bet i*) *then 1 else 0*) $-$ *price i*)
$+$ ($\sum$ *i* $\leftarrow$ *asks strategy*. *price i* $-$ (*if s* (*bet i*) *then 1 else 0*))

Alternate definitions of the payout function π are to use the notion of *settling* bets given a state of affairs. Settling is just paying out those bets that came true.

definition *settle-bet* :: ($'p \Rightarrow bool$) $\Rightarrow$ $'p \Rightarrow$ *real* **where**
settle-bet s $\varphi \equiv$ *if* (*s* φ) *then 1 else 0*

lemma *payoff-alt-def1*:
π *s strategy* $=$
($\sum$ *i* $\leftarrow$ *bids strategy*. *settle-bet s* (*bet i*) $-$ *price i*)
$+$ ($\sum$ *i* $\leftarrow$ *asks strategy*. *price i* $-$ *settle-bet s* (*bet i*))
unfolding *settle-bet-def*
by *simp*

definition *settle* :: ($'p \Rightarrow bool$) $\Rightarrow$ $'p$ *bet-offer list* $\Rightarrow$ *real* **where**

settle s bets ≡ $\sum$ *b* ← *bets*. *settle-bet s* (*bet b*)

lemma *settle-alt-def*:
settle q bets = *length* [φ ← [*bet b* . *b* ← *bets*] . *q* φ]
unfolding *settle-def settle-bet-def*
by (*induct bets*, *simp*+)

definition *total-price* :: ('*p bet-offer*) *list* ⇒ *real* **where**
total-price offers ≡ $\sum$ *i* ← *offers*. *price i*

lemma *payoff-alt-def2*:
π *s strategy* = *settle s* (*bids strategy*)
− *settle s* (*asks strategy*)
+ *total-price* (*asks strategy*)
− *total-price* (*bids strategy*)
unfolding *payoff-alt-def1 total-price-def settle-def*
by (*simp add*: *sum-list-subtractf*)

2.4 Revenue Equivalence

When evaluating a payout function, we can essentially convert *BID* orders to *ASK* orders in a strategy, provided we properly account for locked capital when calculating the effective prices for the new *ASK* positions.

definition (**in** *classical-logic*) *negate-bets* (-$^\sim$) **where**
bets$^\sim$ = [*b* ⦇ *bet* := ∼ (*bet b*) ⦈. *b* ← *bets*]

lemma (**in** *classical-logic*) *ask-revenue-equivalence*:
assumes *possibility p*
shows π *p* ⦇ *asks* = *asks'*, *bids* = *bids'* ⦈
= − *settle p* (*bids'*$^\sim$ @ *asks'*)
+ *total-price asks'*
+ *length bids'*
− *total-price bids'*
proof (*induct bids'*)
case *Nil*
then show *?case*
unfolding
payoff-alt-def2
negate-bets-def
total-price-def
settle-def
by *simp*
next
case (*Cons bid' bids'*)
have *p* (∼ (*bet bid'*)) = (¬ (*p* (*bet bid'*)))
using *assms negation-def* **by** *auto*
moreover have
total-price ((*bid'* # *bids'*) @ *asks'*)
= *price bid'* + *total-price bids'* + *total-price asks'*

unfolding *total-price-def*
by (*induct asks'*, *induct bids'*, *auto*)
ultimately show *?case*
using *Cons*
unfolding *payoff-alt-def2 negate-bets-def settle-def settle-bet-def*
by *simp*
qed

The dual is also true: when evaluating a payout function, we can, in a sense, treat *ASK* as *BID* positions with proper accounting.

lemma (**in** *classical-logic*) *bid-revenue-equivalence*:
assumes *possibility p*
shows $\pi\ p\ (\!|\ asks = asks',\ bids = bids'\ |\!)$
$=\ settle\ p\ (asks'^{\sim}\ @\ bids')$
$+\ total\text{-}price\ asks'$
$-\ total\text{-}price\ bids'$
$-\ length\ asks'$
proof (*induct asks'*)
case *Nil*
then show *?case*
unfolding
payoff-alt-def2
negate-bets-def
total-price-def
settle-def
settle-bet-def
by *simp*
next
case (*Cons s asks'*)
have $p\ (\sim (bet\ s)) = (\neg\ (p\ (bet\ s)))$ **using** *assms negation-def* **by** *auto*
moreover have $total\text{-}price\ ((s\ \#\ asks')\ @\ bids')$
$=\ price\ s + total\text{-}price\ asks' + total\text{-}price\ bids'$
unfolding *total-price-def*
by (*induct bids'*, *induct asks'*, *auto*)
ultimately show *?case*
using *Cons*
unfolding *payoff-alt-def2 negate-bets-def settle-def settle-bet-def*
by *simp*
qed

3 Arbitrage Strategies

3.1 Introduction

In this section, we consider the problem of computing whether a strategy will always yield a profit. Such strategies are referred to as *arbitrage strategies*.

3.2 Minimum Payoff

When computing whether a strategy is suited to arbitrage trading, we need to know the *minimum payoff* of that strategy given every possible scenario.

definition (**in** *consistent-classical-logic*)
minimum-payoff :: *'a strategy* ⇒ *real* (π_{min}) **where**
π_{min} *b* ≡ *THE x*. (∃ *p* ∈ *possibilities*. π *p b* = *x*)
∧ (∀ *q* ∈ *possibilities*. *x* ≤ π *q b*)

Since our definition of π_{min} relies on a definite descriptor, we need the following theorem to prove it is well-defined.

lemma (**in** *consistent-classical-logic*) *minimum-payoff-existence*:
∃! *x*. (∃ *p* ∈ *possibilities*. π *p bets* = *x*) ∧ (∀ *q* ∈ *possibilities*. *x* ≤ π *q bets*)
proof (*rule ex-ex1I*)
show ∃ *x*. (∃ *p*∈*possibilities*. π *p bets* = *x*) ∧ (∀ *q*∈*possibilities*. *x* ≤ π *q bets*)
proof (*rule ccontr*)
obtain *bids′ asks′* **where** *bets* = ⦇ *asks* = *asks′*, *bids* = *bids′* ⦈
by (*metis strategy.cases*)
assume ∄ *x*. (∃ *p* ∈ *possibilities*. π *p bets* = *x*) ∧ (∀ *q* ∈ *possibilities*. *x* ≤ π *q* *bets*)
hence ∀ *x*. (∃ *p* ∈ *possibilities*. π *p bets* = *x*) ⟶ (∃ *q* ∈ *possibilities*. π *q bets* < *x*)
by (*meson le-less-linear*)
hence ⋆: ∀ *p* ∈ *possibilities*. ∃ *q* ∈ *possibilities*. π *q bets* < π *p bets*
by *blast*
have ◊: ∀ *p* ∈ *possibilities*. ∃ *q* ∈ *possibilities*.
settle q (*asks′*$^{\sim}$ @ *bids′*) < *settle p* (*asks′*$^{\sim}$ @ *bids′*)
proof
fix *p*
assume *p* ∈ *possibilities*
from *this* **obtain** *q* **where** *q* ∈ *possibilities* **and** π *q bets* < π *p bets*
using ⋆ **by** *blast*
hence
settle q (*asks′*$^{\sim}$ @ *bids′*)
\+ *total-price asks′*
− *total-price bids′*
− *length asks′*
< *settle p* (*asks′*$^{\sim}$ @ *bids′*)
\+ *total-price asks′*
− *total-price bids′*
− *length asks′*
by (*metis* ‹π *q bets* < π *p bets*›
‹*bets* = ⦇*asks* = *asks′*, *bids* = *bids′*⦈›
‹*p* ∈ *possibilities*›
possibilities-def
bid-revenue-equivalence
mem-Collect-eq)
hence *settle q* (*asks′*$^{\sim}$ @ *bids′*) < *settle p* (*asks′*$^{\sim}$ @ *bids′*)
by *simp*

```
  thus ∃ q∈possibilities. settle q (asks'~ @ bids') < settle p (asks'~ @ bids')
    using ‹q ∈ possibilities› by blast
qed
{
  fix bets :: ('a bet-offer) list
  fix s :: 'a ⇒ bool
  have ∃ n ∈ ℕ. settle s bets = real n
    unfolding settle-def settle-bet-def
    by (induct bets, auto, metis Nats-1 Nats-add Suc-eq-plus1-left of-nat-Suc)
} note † = this
{
  fix n :: nat
  have     (∃ p ∈ possibilities. settle p (asks'~ @ bids') ≤ n)
      ⟶ (∃ q ∈ possibilities. settle q (asks'~ @ bids') < 0)
      (is - ⟶ ?consequent)
  proof (induct n)
    case 0
    {
      fix p :: 'a ⇒ bool
      assume p ∈ possibilities and settle p (asks'~ @ bids') ≤ 0
      from this obtain q where
        q ∈ possibilities
        settle q (asks'~ @ bids') < settle p (asks'~ @ bids')
        using ◊ by blast
      hence ?consequent
        by (metis
              †
              ‹settle p (asks'~ @ bids') ≤ 0›
              of-nat-0-eq-iff
              of-nat-le-0-iff)
    }
    then show ?case by auto
  next
    case (Suc n)
    {
      fix p :: 'a ⇒ bool
      assumep ∈ possibilities and settle p (asks'~ @ bids') ≤ Suc n
      from this obtain q1 where
        q1 ∈ possibilities
        settle q1 (asks'~ @ bids') < Suc n
        by (metis ◊ antisym-conv not-less)
      from this obtain q2 where
        q2 ∈ possibilities
        settle q2 (asks'~ @ bids') < n
        using ◊
        by (metis
              †
              add.commute
              nat-le-real-less
```

```
          nat-less-le
          of-nat-Suc
          of-nat-less-iff)
      hence ?consequent
        by (metis † Suc.hyps nat-less-le of-nat-le-iff of-nat-less-iff)
    }
    then show ?case by auto
  qed
 }
 hence ∄ p. p ∈ possibilities
   by (metis † not-less0 of-nat-0 of-nat-less-iff order-refl)
 moreover
 have ¬ {} ⊩ ⊥
   using consistency set-deduction-base-theory by auto
 from this obtain Γ where MCS Γ
   by (meson formula-consistent-def
             formula-maximal-consistency
             formula-maximally-consistent-extension)
 hence (λ γ. γ ∈ Γ) ∈ possibilities
   using MCSs-are-possibilities possibilities-def by blast
 ultimately show False
   by blast
 qed
next
 fix x y
 assume A: (∃ p ∈ possibilities. π p bets = x) ∧ (∀ q ∈ possibilities. x ≤ π q bets)
 and B: (∃ p ∈ possibilities. π p bets = y) ∧ (∀ q ∈ possibilities. y ≤ π q bets)
 from this obtain p_x p_y where
   p_x ∈ possibilities
   p_y ∈ possibilities
   π p_x bets = x
   π p_y bets = y
   by blast
 with A B have x ≤ y y ≤ x
   by blast+
 thus x = y by linarith
qed
```

3.3 Bounding Minimum Payoffs Below Using MaxSAT

Below, we present our second major theorem: computing a lower bound to a strategy's minimum payoff is equivalent to checking a bounded MaxSAT problem.

A concrete implementation of this algorithm would enable software search for trading strategies that can convert orders from one central limit order book to another.

As in the previous section, we prove our theorem in the general case of an arbitrary k, but in practice users will want to set $k = 0$ to check if their strategy is an arbitrage strategy.

```
theorem (in consistent-classical-logic) arbitrageur-maxsat:
  ((k :: real) ≤ π_min ⦇ asks = asks', bids = bids' ⦈)
 = (  MaxSAT [bet b . b ← bids'~ @ asks']
     ≤ total-price asks' + length bids' − total-price bids' − k )
 (is (k ≤ π_min ?bets) = (MaxSAT ?props ≤ total-price - + - − - − -))
proof
 assume k ≤ π_min ?bets
 let ?P = λ x . (∃ p ∈ possibilities. π p ?bets = x)
                  ∧ (∀ q ∈ possibilities. x ≤ π q ?bets)
 obtain p where
     possibility p and
     ∀ q ∈ possibilities. π p ?bets ≤ π q ?bets
   using ‹k ≤ π_min ?bets›
         minimum-payoff-existence [of ?bets]
   by (metis possibilities-def mem-Collect-eq)
 hence ?P (π p ?bets)
   using possibilities-def by blast
 hence π_min ?bets = π p ?bets
   unfolding minimum-payoff-def
   using minimum-payoff-existence [of ?bets]
         the1-equality [where P = ?P and a = π p ?bets]
   by blast

 let ?Φ = [φ ← ?props. p φ]

 have mset ?Φ ⊆# mset ?props
   by(induct ?props,
      auto,
      simp add: subset-mset.add-mono)
 moreover
 have ¬ (?Φ :⊢ ⊥)
 proof −
   have set ?Φ ⊆ {x. p x}
     by auto
   hence ¬ (set ?Φ ⊩ ⊥)
     by (meson ‹possibility p›
               possibilities-are-MCS [of p]
               formula-consistent-def
               formula-maximally-consistent-set-def-def
               maximally-consistent-set-def
               list-deduction-monotonic
               set-deduction-def)
   thus ?thesis
     using set-deduction-def by blast
 qed
 moreover
 {
   fix Ψ
```

```
assume mset Ψ ⊆# mset ?props and ¬ Ψ :⊢ ⊥
from this obtain Ω_Ψ where MCS Ω_Ψ and set Ψ ⊆ Ω_Ψ
  by (meson formula-consistent-def
            formula-maximal-consistency
            formula-maximally-consistent-extension
            list-deduction-monotonic
            set-deduction-def)
let ?q = λφ . φ ∈ Ω_Ψ
have possibility ?q
  using ‹MCS Ω_Ψ› MCSs-are-possibilities by blast
hence π p ?bets ≤ π ?q ?bets
  using ‹∀ q∈possibilities. π p ?bets ≤ π q ?bets›
        possibilities-def
  by blast
let ?c = total-price asks' + length bids' − total-price bids'
have − settle p (bids'~ @ asks') + ?c ≤ − settle ?q (bids'~ @ asks') + ?c
  using ‹π p ?bets ≤ π ?q ?bets›
        ‹possibility p›
        ask-revenue-equivalence [of p asks' bids']
        ‹possibility ?q›
        ask-revenue-equivalence [of ?q asks' bids']
  by linarith
hence settle ?q (bids'~ @ asks') ≤ settle p (bids'~ @ asks')
  by linarith
let ?Ψ' = [φ ← ?props. ?q φ]
have length ?Ψ' ≤ length ?Φ
  using ‹settle ?q (bids'~ @ asks') ≤ settle p (bids'~ @ asks')›
  unfolding settle-alt-def
  by simp
moreover
have length Ψ ≤ length ?Ψ'
proof −
  have mset [ψ ← Ψ. ?q ψ] ⊆# mset ?Ψ'
  proof −
    {
      fix props :: 'a list
      have ∀ Ψ. ∀ Ω. mset Ψ ⊆# mset props ⟶
                    mset [ψ ← Ψ. ψ ∈ Ω] ⊆# mset [φ ← props. φ ∈ Ω]
        by (simp add: multiset-filter-mono)
    }
    thus ?thesis
      using ‹mset Ψ ⊆# mset ?props› by blast
  qed
  hence length [ψ ← Ψ. ?q ψ] ≤ length ?Ψ'
   by (metis (no-types, lifting) length-sub-mset mset-eq-length nat-less-le not-le)
  moreover have length Ψ = length [ψ ← Ψ. ?q ψ]
    using ‹set Ψ ⊆ Ω_Ψ›
    by (induct Ψ, simp+)
  ultimately show ?thesis by linarith
```

```
    qed
    ultimately have length Ψ ≤ length ?Φ by linarith
  }
  ultimately have ?Φ ∈ ℳ ?props ⊥
    unfolding relative-maximals-def
    by blast
  hence MaxSAT ?props = length ?Φ
    using relative-MaxSAT-intro by presburger
  hence MaxSAT ?props = settle p (bids'~ @ asks')
    unfolding settle-alt-def
    by simp
  thus MaxSAT ?props ≤ total-price asks' + length bids' − total-price bids' − k
    using ask-revenue-equivalence [of p asks' bids']
          ‹k ≤ π_min ?bets›
          ‹π_min ?bets = π p ?bets›
          ‹possibility p›
    by linarith
next
  let ?c = total-price asks' + length bids' − total-price bids'
  assume MaxSAT ?props ≤ total-price asks' + length bids' − total-price bids' −
k
  from this obtain Φ where Φ ∈ ℳ ?props ⊥ and length Φ + k ≤ ?c
    using
      consistency
      relative-MaxSAT-intro
      relative-maximals-existence
    by fastforce
  hence ¬ Φ :⊢ ⊥
    using relative-maximals-def by blast
  from this obtain Ω_Φ where MCS Ω_Φ and set Φ ⊆ Ω_Φ
    by (meson formula-consistent-def
              formula-maximal-consistency
              formula-maximally-consistent-extension
              list-deduction-monotonic
              set-deduction-def)
  let ?p = λφ . φ ∈ Ω_Φ
  have possibility ?p
    using ‹MCS Ω_Φ› MCSs-are-possibilities by blast
  have mset Φ ⊆# mset ?props
    using ‹Φ ∈ ℳ ?props ⊥› relative-maximals-def by blast
  have mset Φ ⊆# mset [ b ← ?props. ?p b]
    by (metis ‹mset Φ ⊆# mset ?props›
              ‹set Φ ⊆ Ω_Φ›
              filter-True
              mset-filter
              multiset-filter-mono
              subset-code(1))
  have mset Φ = mset [ b ← ?props. ?p b]
  proof (rule ccontr)
```

```
  assume mset Φ ≠ mset [ b ← ?props. ?p b]
  hence length Φ < length [ b ← ?props. ?p b]
    using
      ‹mset Φ ⊆# mset [ b ← ?props. ?p b]›
      length-sub-mset not-less
    by blast
  moreover
  have ¬ [ b ← ?props. ?p b] :⊢ ⊥
    by (metis
          IntE
          ‹MCS Ω_Φ›
          inter-set-filter
          formula-consistent-def
          formula-maximally-consistent-set-def-def
          maximally-consistent-set-def
          set-deduction-def
          subsetI)
  hence length [ b ← ?props. ?p b] ≤ length Φ
    by (metis
          (mono-tags, lifting)
          ‹Φ ∈ ℳ ?props ⊥›
          relative-maximals-def [of ?props ⊥]
          mem-Collect-eq
          mset-filter
          multiset-filter-subset)
  ultimately show False
    using not-le by blast
qed
hence length Φ = settle ?p (bids'~ @ asks')
  unfolding settle-alt-def
  using mset-eq-length
  by metis
hence k ≤ settle ?p (bids'~ @ asks')
          + total-price asks' + length bids' − total-price bids'
  using ‹length Φ + k ≤ ?c› by linarith
hence k ≤ π ?p ?bets
  using ‹possibility ?p›
        ask-revenue-equivalence [of ?p asks' bids']
        ‹length Φ + k ≤ ?c›
        ‹length Φ = settle ?p (bids'~ @ asks')›
  by linarith
have ∀ q ∈ possibilities. π ?p ?bets ≤ π q ?bets
proof
  {
    fix x :: 'a
    fix P A
    have x ∈ Set.filter P A ⟷ x ∈ A ∧ P x
      by (simp add: filter-def)
  }
```

```
  note member-filter = this
  fix q
  assume q ∈ possibilities
  hence ¬ [ b ← ?props. q b] :⊢ ⊥
    unfolding possibilities-def
    by (metis filter-set
              possibilities-logical-closure
              possibility-def
              set-deduction-def
              mem-Collect-eq
              member-filter
              subsetI)
  hence length [ b ← ?props. q b] ≤ length Φ
    by (metis (mono-tags, lifting)
              ‹Φ ∈ ℳ ?props ⊥›
              relative-maximals-def
              mem-Collect-eq
              mset-filter
              multiset-filter-subset)
  hence
      − settle ?p (bids'~ @ asks')
        + total-price asks'
        + length bids'
        − total-price bids'
    ≤ − settle q (bids'~ @ asks')
        + total-price asks'
        + length bids'
        − total-price bids'
    using ‹length Φ = settle ?p (bids'~ @ asks')›
          settle-alt-def [of q bids'~ @ asks']
    by linarith
  thus π ?p ?bets ≤ π q ?bets
    using ask-revenue-equivalence [of ?p asks' bids']
          ask-revenue-equivalence [of q asks' bids']
          ‹possibility ?p›
          ‹q ∈ possibilities›
    unfolding possibilities-def
    by (metis mem-Collect-eq)
qed
have π_min ?bets = π ?p ?bets
  unfolding minimum-payoff-def
proof
  show (∃ p∈possibilities. π p ?bets = π ?p ?bets)
          ∧ (∀ q∈possibilities. π ?p ?bets ≤ π q ?bets)
    using ‹∀ q ∈ possibilities. π ?p ?bets ≤ π q ?bets›
          ‹possibility ?p›
    unfolding possibilities-def
    by blast
next
```

```
    fix n
    assume ⋆: (∃ p∈possibilities. π p ?bets = n) ∧ (∀ q∈possibilities. n ≤ π q ?bets)
    from this obtain p where π p ?bets = n and possibility p
      using possibilities-def by blast
    hence π p ?bets ≤ π ?p ?bets
      using ⋆ ‹possibility ?p›
      unfolding possibilities-def
      by blast
    moreover have π ?p ?bets ≤ π p ?bets
      using ‹∀ q ∈ possibilities. π ?p ?bets ≤ π q ?bets›
            ‹possibility p›
      unfolding possibilities-def
      by blast
    ultimately show n = π ?p ?bets using ‹π p ?bets = n› by linarith
  qed
  thus k ≤ π_min ?bets
    using ‹k ≤ π ?p ?bets›
    by auto
qed
```

4 Coherence Checking

4.1 Introduction

In this section, we give an abstract algorithm for traders to use to detect if a strategy they want to employ will *always lose*, i.e., is *incoherent*.

4.2 Maximum Payoff

The key to figuring out if a trading strategy will not always lose is computing the strategy's *maximum payoff*.

Below, we define the maximum payoff using a definite description.

```
definition (in consistent-classical-logic)
  maximum-payoff :: 'a strategy ⇒ real (π_max) where
  π_max b ≡ THE x. (∃ p ∈ possibilities. π p b = x)
                 ∧ (∀ q ∈ possibilities. π q b ≤ x)
```

The following lemma establishes that our definition of π_{max} is well-defined.

```
lemma (in consistent-classical-logic) maximum-payoff-existence:
  ∃! x. (∃ p ∈ possibilities. π p bets = x)
       ∧ (∀ q ∈ possibilities. π q bets ≤ x)
proof (rule ex-ex1I)
  show ∃ x. (∃ p ∈possibilities. π p bets = x)
          ∧ (∀ q ∈ possibilities. π q bets ≤ x)
  proof (rule ccontr)
    obtain bids' asks' where bets = (| asks = asks', bids = bids' |)
      by (metis strategy.cases)
```

```
assume ∄x. (∃p∈possibilities. π p bets = x)
             ∧ (∀q∈possibilities. π q bets ≤ x)
hence ∀x. (∃ p ∈ possibilities. π p bets = x)
            ⟶ (∃ q ∈ possibilities. x < π q bets)
  by (meson le-less-linear)
hence ⋆: ∀p ∈ possibilities. ∃ q ∈ possibilities. π p bets < π q bets
  by blast
have ◊: ∀ p ∈ possibilities. ∃ q ∈ possibilities.
              settle p (asks'~ @ bids') < settle q (asks'~ @ bids')
proof
  fix p
  assume p ∈ possibilities
  from this obtain q where q ∈ possibilities and π p bets < π q bets
    using ⋆ by blast
  hence
        settle p (asks'~ @ bids')
      + total-price asks'
      − total-price bids'
      − length asks'
    <   settle q (asks'~ @ bids')
      + total-price asks'
      − total-price bids'
      − length asks'
    by (metis ‹π p bets < π q bets›
            ‹bets = (|asks = asks', bids = bids'|)›
            ‹p ∈ possibilities›
            possibilities-def
            bid-revenue-equivalence
            mem-Collect-eq)
  hence settle p (asks'~ @ bids') < settle q (asks'~ @ bids')
    by simp
  thus ∃q∈possibilities.   settle p (asks'~ @ bids')
                          < settle q (asks'~ @ bids')
    using ‹q ∈ possibilities› by blast
qed
{
  fix bets :: ('a bet-offer) list
  fix s :: 'a ⇒ bool
  have ∃ n ∈ ℕ. settle s bets = real n
    unfolding settle-def settle-bet-def
    by (induct bets,
        auto,
        metis
          Nats-1
          Nats-add
          Suc-eq-plus1-left of-nat-Suc)
} note † = this
{
  fix n :: nat
```

```
have ∃ q ∈ possibilities. n ≤ settle q (asks'~ @ bids')
  by (induct n,
      metis
        †
        MCSs-are-possibilities
        consistency
        formula-consistent-def
        formula-maximal-consistency
        formula-maximally-consistent-extension
        possibilities-def
        set-deduction-base-theory
        mem-Collect-eq
        of-nat-0
        of-nat-0-le-iff,
      metis ◊ † le-antisym not-less not-less-eq-eq of-nat-less-iff)
  }
  moreover
  {
    fix bets :: ('a bet-offer) list
    fix s :: 'a ⇒ bool
    have settle s bets ≤ length bets
      unfolding settle-def settle-bet-def
      by (induct bets, auto)
  }
  ultimately show False
    by (metis † not-less-eq-eq of-nat-le-iff)
 qed
next
 fix x y
 assume A: (∃ p∈possibilities. π p bets = x) ∧ (∀ q∈possibilities. π q bets ≤ x)
 and B: (∃ p∈possibilities. π p bets = y) ∧ (∀ q∈possibilities. π q bets ≤ y)
 from this obtain p_x p_y where
   p_x ∈ possibilities
   p_y ∈ possibilities
   π p_x bets = x
   π p_y bets = y
   by blast
 with A B have x ≤ y y ≤ x
   by blast+
 thus x = y by linarith
qed
```

4.3 Bounding Maximum Payoffs Above Using MaxSAT

Below, we present our first major theorem: computing an upper bound to a strategy's maximum payoff is equivalent to a bounded MaxSAT problem.

Given a software MaxSAT implementation, a trader can use this equivalence to run a program to check whether the way they arrive at their strategies has a bug.

Note that while the theorem below is formulated using an arbitrary k constant, in practice users will want to check their strategies are safe by using $k = 0$.

```
theorem (in consistent-classical-logic) coherence-maxsat:
  (π_max (| asks = asks', bids = bids' |) ≤ (k :: real))
  = (MaxSAT [bet b . b ← asks'~ @ bids']
      ≤ k − total-price asks' + total-price bids' + length asks')
 (is (π_max ?bets ≤ k) = (MaxSAT ?props ≤ - − total-price - + - + -))
proof
 assume π_max ?bets ≤ k
 let ?P = λ x . (∃ p ∈ possibilities. π p ?bets = x)
                   ∧ (∀ q ∈ possibilities. π q ?bets ≤ x)
 obtain p where
    possibility p and
    ∀ q ∈ possibilities. π q ?bets ≤ π p ?bets
   using ‹π_max ?bets ≤ k›
        maximum-payoff-existence [of ?bets]
   by (metis possibilities-def mem-Collect-eq)
 hence ?P (π p ?bets)
   using possibilities-def by blast
 hence π_max ?bets = π p ?bets
   unfolding maximum-payoff-def
   using maximum-payoff-existence [of ?bets]
        the1-equality [where P = ?P and a = π p ?bets]
   by blast

 let ?Φ = [φ ← ?props. p φ]

 have mset ?Φ ⊆# mset ?props
   by(induct ?props,
      auto,
      simp add: subset-mset.add-mono)
 moreover
 have ¬ (?Φ :⊢ ⊥)
 proof −
   have set ?Φ ⊆ {x. p x}
     by auto
   hence ¬ (set ?Φ ⊩ ⊥)
     by (meson
          ‹possibility p›
          possibilities-are-MCS [of p]
          formula-consistent-def
          formula-maximally-consistent-set-def-def
          maximally-consistent-set-def
          list-deduction-monotonic
          set-deduction-def)
   thus ?thesis
```

```
    using set-deduction-def by blast
qed
moreover
{
  fix Ψ
  assume mset Ψ ⊆# mset ?props and ¬ Ψ :⊢ ⊥
  from this obtain Ω_Ψ where MCS Ω_Ψ and set Ψ ⊆ Ω_Ψ
    by (meson
          formula-consistent-def
          formula-maximal-consistency
          formula-maximally-consistent-extension
          list-deduction-monotonic
          set-deduction-def)
  let ?q = λφ . φ ∈ Ω_Ψ
  have possibility ?q
    using ‹MCS Ω_Ψ› MCSs-are-possibilities by blast
  hence π ?q ?bets ≤ π p ?bets
    using ‹∀ q∈possibilities. π q ?bets ≤ π p ?bets›
          possibilities-def
    by blast
  let ?c = total-price asks' − total-price bids' − length asks'
  have settle ?q (asks'~ @ bids') + ?c ≤ settle p (asks'~ @ bids') + ?c
    using ‹π ?q ?bets ≤ π p ?bets›
          ‹possibility p›
          bid-revenue-equivalence [of p asks' bids']
          ‹possibility ?q›
          bid-revenue-equivalence [of ?q asks' bids']
    by linarith
  hence settle ?q (asks'~ @ bids') ≤ settle p (asks'~ @ bids')
    by linarith
  let ?Ψ' = [φ ← ?props. ?q φ]
  have length ?Ψ' ≤ length ?Φ
    using ‹settle ?q (asks'~ @ bids') ≤ settle p (asks'~ @ bids')›
    unfolding settle-alt-def
    by simp
  moreover
  have length Ψ ≤ length ?Ψ'
  proof −
    have mset [ψ ← Ψ. ?q ψ] ⊆# mset ?Ψ'
    proof −
      {
        fix props :: 'a list
        have ∀ Ψ. ∀ Ω.
                mset Ψ ⊆# mset props
                  ⟶ mset [ψ ← Ψ. ψ ∈ Ω] ⊆# mset [φ ← props. φ ∈ Ω]
          by (simp add: multiset-filter-mono)
      }
      thus ?thesis
        using ‹mset Ψ ⊆# mset ?props› by blast
```

qed
hence *length* $[\psi \leftarrow \Psi.\ ?q\ \psi] \leq$ *length* $?\Psi'$
by (*metis*
(*no-types*, *lifting*)
length-sub-mset
mset-eq-length
nat-less-le
not-le)
moreover have *length* $\Psi =$ *length* $[\psi \leftarrow \Psi.\ ?q\ \psi]$
using ‹*set* $\Psi \subseteq \Omega_\Psi$›
by (*induct* Ψ, *simp*+)
ultimately show *?thesis* **by** *linarith*
qed
ultimately have *length* $\Psi \leq$ *length* $?\Phi$ **by** *linarith*
}
ultimately have $?\Phi \in \mathcal{M}$ *?props* $\bot$
unfolding *relative-maximals-def*
by *blast*
hence *MaxSAT ?props* = *length* $?\Phi$
using *relative-MaxSAT-intro* **by** *presburger*
hence *MaxSAT ?props* = *settle p* (*asks*$'^{\sim}$ @ *bids*$'$)
unfolding *settle-alt-def*
by *simp*
thus *MaxSAT ?props*
$\leq k -$ *total-price asks*$'$ + *total-price bids*$'$ + *length asks*$'$
using *bid-revenue-equivalence* [*of p asks*$'$ *bids*$'$]
‹π_{max} *?bets* $\leq k$›
‹π_{max} *?bets* $= \pi$ *p ?bets*›
‹*possibility p*›
by *linarith*
next
let *?c* = − *total-price asks*$'$ + *total-price bids*$'$ + *length asks*$'$
assume *MaxSAT ?props*
$\leq k -$ *total-price asks*$'$ + *total-price bids*$'$ + *length asks*$'$
from *this* **obtain** Φ **where** $\Phi \in \mathcal{M}$ *?props* $\bot$ **and** *length* $\Phi \leq k +$ *?c*
using
consistency
relative-MaxSAT-intro
relative-maximals-existence
by *fastforce*
hence $\neg\ \Phi :\vdash \bot$
using *relative-maximals-def* **by** *blast*
from *this* **obtain** Ω_Φ **where** *MCS* Ω_Φ **and** *set* $\Phi \subseteq \Omega_\Phi$
by (*meson*
formula-consistent-def
formula-maximal-consistency
formula-maximally-consistent-extension
list-deduction-monotonic
set-deduction-def)

let *?p* = $\lambda\varphi$. $\varphi \in \Omega_\Phi$
have *possibility ?p*
  using ‹*MCS* Ω_Φ› *MCSs-are-possibilities* **by** *blast*
have *mset* Φ ⊆# *mset ?props*
  using ‹$\Phi \in \mathcal{M}$ *?props* ⊥› *relative-maximals-def* **by** *blast*
have *mset* Φ ⊆# *mset* [*b* ← *?props*. *?p b*]
  by (*metis*
      ‹*mset* Φ ⊆# *mset ?props*›
      ‹*set* $\Phi \subseteq \Omega_\Phi$›
      filter-True
      mset-filter
      multiset-filter-mono
      subset-code(*1*))
have *mset* Φ = *mset* [*b* ← *?props*. *?p b*]
proof (*rule ccontr*)
  assume *mset* $\Phi \neq$ *mset* [*b* ← *?props*. *?p b*]
  hence *length* Φ < *length* [*b* ← *?props*. *?p b*]
    using
      ‹*mset* Φ ⊆# *mset* [*b* ← *?props*. *?p b*]›
      length-sub-mset not-less
    by *blast*
  moreover
  have ¬ [*b* ← *?props*. *?p b*] :⊢ ⊥
    by (*metis*
        IntE
        ‹*MCS* Ω_Φ›
        inter-set-filter
        formula-consistent-def
        formula-maximally-consistent-set-def-def
        maximally-consistent-set-def
        set-deduction-def
        subsetI)
  hence *length* [*b* ← *?props*. *?p b*] ≤ *length* Φ
    by (*metis*
        (*mono-tags*, *lifting*)
        ‹$\Phi \in \mathcal{M}$ *?props* ⊥›
        relative-maximals-def [*of ?props* ⊥]
        mem-Collect-eq
        mset-filter
        multiset-filter-subset)
  ultimately show *False*
    using *not-le* **by** *blast*
qed
hence *length* Φ = *settle ?p* (*asks*$'^\sim$ @ *bids*$'$)
  unfolding *settle-alt-def*
  using *mset-eq-length*
  by *metis*
hence *settle ?p* (*asks*$'^\sim$ @ *bids*$'$) ≤ *k* + *?c*
  using ‹*length* Φ ≤ *k* + *?c*› **by** *linarith*

```
hence π ?p ?bets ≤ k
  using ‹possibility ?p›
        bid-revenue-equivalence [of ?p asks' bids']
        ‹length Φ ≤ k + ?c›
        ‹length Φ = settle ?p (asks'~ @ bids')›
  by linarith
have ∀ q ∈ possibilities. π q ?bets ≤ π ?p ?bets
proof
  {
    fix x :: 'a
    fix P A
    have x ∈ Set.filter P A ⟷ x ∈ A ∧ P x
      by (simp add: filter-def)
  }
  note member-filter = this
  fix q
  assume q ∈ possibilities
  hence possibility q unfolding possibilities-def by auto
  hence ¬ [ b ← ?props. q b] :⊢ ⊥
    by (metis filter-set
              possibilities-logical-closure
              possibility-def
              set-deduction-def
              mem-Collect-eq
              member-filter
              subsetI)
  hence length [ b ← ?props. q b] ≤ length Φ
    by (metis (mono-tags, lifting)
              ‹Φ ∈ ℳ ?props ⊥›
              relative-maximals-def
              mem-Collect-eq
              mset-filter
              multiset-filter-subset)
  hence settle q (asks'~ @ bids') ≤ length Φ
    by (metis of-nat-le-iff settle-alt-def)
  thus π q ?bets ≤ π ?p ?bets
    using bid-revenue-equivalence [OF ‹possibility ?p›]
          bid-revenue-equivalence [OF ‹possibility q›]
          ‹length Φ = settle ?p (asks'~ @ bids')›
    by force
qed
have π_max ?bets = π ?p ?bets
  unfolding maximum-payoff-def
proof
  show (∃ p∈possibilities. π p ?bets = π ?p ?bets)
       ∧ (∀ q∈possibilities. π q ?bets ≤ π ?p ?bets)
    using ‹∀ q ∈ possibilities. π q ?bets ≤ π ?p ?bets›
          ‹possibility ?p›
    unfolding possibilities-def
```

by *blast*
next
fix *n*
assume ⋆: $(\exists\, p \in possibilities.\ \pi\ p\ ?bets = n)$
$\wedge\ (\forall\, q \in possibilities.\ \pi\ q\ ?bets \leq n)$
from *this* **obtain** *p* **where** $\pi\ p\ ?bets = n$ **and** *possibility p*
using *possibilities-def* **by** *blast*
hence $\pi\ ?p\ ?bets \leq \pi\ p\ ?bets$
using ⋆ ‹*possibility ?p*›
unfolding *possibilities-def*
by *blast*
moreover have $\pi\ p\ ?bets \leq \pi\ ?p\ ?bets$
using ‹$\forall\ q \in possibilities.\ \pi\ q\ ?bets \leq \pi\ ?p\ ?bets$›
‹*possibility p*›
unfolding *possibilities-def*
by *blast*
ultimately show $n = \pi\ ?p\ ?bets$ **using** ‹$\pi\ p\ ?bets = n$› **by** *linarith*
qed
thus $\pi_{max}\ ?bets \leq k$
using ‹$\pi\ ?p\ ?bets \leq k$›
by *auto*
qed

5 Probability Inequality Identity Correspondence

5.1 Introduction

In this section, we prove two forms of the probability inequality identity correspondence theorem.

The two forms relate to π_{min} (i.e., arbitrage strategy determination) and π_{max} (i.e., coherence testing).

In each case, the form follows from the reduction to bounded MaxSAT previously presented, and the reduction of bounded MaxSAT to probability logic, we established in *Probability-Inequality-Completeness.Probability-Inequality-Completeness*.

5.2 Arbitrage Strategies and Minimum Payoff

First, we connect checking if a strategy is an arbitrage strategy and probability identities.

lemma (**in** *consistent-classical-logic*) *arbitrageur-nonstrict-correspondence*:
$(k \leq \pi_{min}\ (\!|\ asks = asks',\ bids = bids'\ |\!))$
$= (\forall\ \mathcal{P} \in probabilities.$
$(\sum b{\leftarrow}asks'.\ \mathcal{P}\ (bet\ b)) + total\text{-}price\ bids' + k$
$\leq (\sum s{\leftarrow}bids'.\ \mathcal{P}\ (bet\ s)) + total\text{-}price\ asks')$
(**is** *?lhs* = -)
proof −

let *?tot-bs* = *total-price bids′* **and** *?tot-ss* = *total-price asks′*
let *?c* = *?tot-bs* − *?tot-ss* + *k*
have [*bet b* . *b* ← *bids′*$^{\sim}$ @ *asks′*] = ∼ [*bet s*. *s* ← *bids′*] @ [*bet b*. *b* ← *asks′*]
(**is** - = ∼ *?bid-φs* @ *?ask-φs*)
unfolding *negate-bets-def*
by (*induct bids′*, *simp*+)
hence
?lhs = (∀ 𝒫 ∈ *dirac-measures*. (∑φ←*?ask-φs*. 𝒫 φ) + *?c* ≤ (∑γ←*?bid-φs*. 𝒫 γ))
using
dirac-inequality-equiv [*of ?ask-φs ?c ?bid-φs*]
arbitrageur-maxsat [*of k asks′ bids′*]
by *force*
moreover
{
fix 𝒫 :: ′*a* ⇒ *real*
have (∑φ←*?ask-φs*. 𝒫 φ) = (∑ *b*←*asks′*. 𝒫 (*bet b*))
(∑γ←*?bid-φs*. 𝒫 γ) = (∑ *s*←*bids′*. 𝒫 (*bet s*))
by (*simp add*: *comp-def*)+
hence ((∑φ←*?ask-φs*. 𝒫 φ) + *?c* ≤ (∑γ←*?bid-φs*. 𝒫 γ))
= ((∑ *b*←*asks′*. 𝒫 (*bet b*)) + *?tot-bs* + *k*
≤ (∑ *s*←*bids′*. 𝒫 (*bet s*)) + *?tot-ss*)
by *linarith*
}
ultimately show *?thesis*
by (*meson dirac-measures-subset dirac-ceiling dirac-collapse subset-eq*)
qed

lemma (**in** *consistent-classical-logic*) *arbitrageur-strict-correspondence*:
(*k* < π_{min} (| *asks* = *asks′*, *bids* = *bids′* |))
= (∀ 𝒫 ∈ *probabilities*.
(∑ *b*←*asks′*. 𝒫 (*bet b*)) + *total-price bids′* + *k*
< (∑ *s*←*bids′*. 𝒫 (*bet s*)) + *total-price asks′*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
from *this* **obtain** ε **where** *0* < ε *k* + ε ≤ π_{min} (|*asks* = *asks′*, *bids* = *bids′*|)
using *less-diff-eq* **by** *fastforce*
hence ∀𝒫 ∈ *probabilities*.
(∑ *b*←*asks′*. 𝒫 (*bet b*)) + *total-price bids′* + (*k* + ε)
≤ (∑ *s*←*bids′*. 𝒫 (*bet s*)) + *total-price asks′*
using *arbitrageur-nonstrict-correspondence* [*of k* + ε *asks′ bids′*] **by** *auto*
thus *?rhs*
using ‹*0* < ε› **by** *auto*
next
have [*bet b* . *b* ← *bids′*$^{\sim}$ @ *asks′*] = ∼ [*bet s*. *s* ← *bids′*] @ [*bet b*. *b* ← *asks′*]
(**is** - = ∼ *?bid-φs* @ *?ask-φs*)
unfolding *negate-bets-def*

```
    by (induct bids', simp+)
  {
    fix 𝒫 :: 'a ⇒ real
    have (∑ b←asks'. 𝒫 (bet b)) = (∑ φ←?ask-φs. 𝒫 φ)
         (∑ b←bids'. 𝒫 (bet b)) = (∑ φ←?bid-φs. 𝒫 φ)
      by (induct asks', auto, induct bids', auto)
  }
  note ⋆ = this
  let ?tot-bs = total-price bids' and ?tot-ss = total-price asks'
  let ?c = ?tot-bs + k − ?tot-ss
  assume ?rhs
  have ∀ 𝒫 ∈ probabilities. (∑ b←asks'. 𝒫 (bet b)) + ?c < (∑ s←bids'. 𝒫 (bet s))
    using ‹?rhs› by fastforce
  hence ∀ 𝒫 ∈ probabilities. (∑ φ←?ask-φs. 𝒫 φ) + ?c < (∑ φ←?bid-φs. 𝒫 φ)
    using ⋆ by auto
  hence ∀ 𝒫 ∈ dirac-measures. (∑ φ←?ask-φs. 𝒫 φ) + (⌊?c⌋ + 1) ≤ (∑ φ←?bid-φs.
𝒫 φ)
    using strict-dirac-collapse [of ?ask-φs ?c ?bid-φs]
    by auto
  hence MaxSAT (∼ ?bid-φs @ ?ask-φs) + (⌊?c⌋ + 1) ≤ length ?bid-φs
    by (metis floor-add-int floor-mono floor-of-nat dirac-inequality-equiv)
  hence MaxSAT (∼ ?bid-φs @ ?ask-φs) + ?c < length ?bid-φs
    by linarith
  from this obtain ε :: real where
    0 < ε
    MaxSAT (∼ ?bid-φs @ ?ask-φs) + (k + ε) ≤ ?tot-ss + length bids' − ?tot-bs
    using less-diff-eq by fastforce
  hence k + ε ≤ π_min (|asks = asks', bids = bids'|)
    using ‹[bet b . b ← bids'~ @ asks'] = ∼ ?bid-φs @ ?ask-φs›
          arbitrageur-maxsat [of k + ε asks' bids']
    by simp
  thus ?lhs
    using ‹0 < ε› by linarith
qed
```

Below is our central result regarding checking if a strategy is an arbitrage strategy:

A strategy is an arbitrage strategy if and only if there is a corresponding identity in probability theory that reflects it.

```
theorem (in consistent-classical-logic) arbitrageur-correspondence:
    (0 < π_min (| asks = asks', bids = bids' |))
   = (∀ 𝒫 ∈ probabilities.
        (∑ b←asks'. 𝒫 (bet b)) + total-price bids'
      < (∑ s←bids'. 𝒫 (bet s)) + total-price asks')
  by (simp add: arbitrageur-strict-correspondence)
```

5.3 Coherence Checking and Maximum Payoff

Finally, we show the connection between coherence checking and probability identities.

```
lemma (in consistent-classical-logic) coherence-nonstrict-correspondence:
    (π_max ⦇ asks = asks′, bids = bids′ ⦈ ≤ k)
  = (∀ 𝒫 ∈ probabilities.
        (∑ b←bids′. 𝒫 (bet b)) + total-price asks′
      ≤ (∑ s←asks′. 𝒫 (bet s)) + total-price bids′ + k)
  (is ?lhs = -)
proof −
  let ?tot-bs = total-price bids′ and ?tot-ss = total-price asks′
  let ?c = ?tot-ss − ?tot-bs − k
  have [bet b . b ← asks′~ @ bids′] = ∼ [bet s. s ← asks′] @ [bet b. b ← bids′]
    (is - = ∼ ?ask-φs @ ?bid-φs)
    unfolding negate-bets-def
    by (induct bids′, simp+)
  hence
     ?lhs = (∀ 𝒫 ∈ dirac-measures. (∑ φ←?bid-φs. 𝒫 φ) + ?c ≤ (∑ γ←?ask-φs.
𝒫 γ))
    using
      dirac-inequality-equiv [of ?bid-φs ?c ?ask-φs]
      coherence-maxsat [of asks′ bids′ k]
    by force
  moreover
  {
    fix 𝒫 :: ′a ⇒ real
    have (∑ φ←?ask-φs. 𝒫 φ) = (∑ b←asks′. 𝒫 (bet b))
         (∑ γ←?bid-φs. 𝒫 γ) = (∑ s←bids′. 𝒫 (bet s))
      by (simp add: comp-def)+
    hence   ((∑ φ←?bid-φs. 𝒫 φ) + ?c ≤ (∑ γ←?ask-φs. 𝒫 γ))
           = ((∑ b←bids′. 𝒫 (bet b)) + ?tot-ss
               ≤ (∑ s←asks′. 𝒫 (bet s)) + ?tot-bs + k)
      by linarith
  }
  ultimately show ?thesis
    by (meson dirac-measures-subset dirac-ceiling dirac-collapse subset-eq)
qed

lemma (in consistent-classical-logic) coherence-strict-correspondence:
    (π_max ⦇ asks = asks′, bids = bids′ ⦈ < k)
  = (∀ 𝒫 ∈ probabilities.
        (∑ b←bids′. 𝒫 (bet b)) + total-price asks′
      < (∑ s←asks′. 𝒫 (bet s)) + total-price bids′ + k)
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  from this obtain ε where 0 < ε π_max ⦇asks = asks′, bids = bids′⦈ + ε ≤ k
    using less-diff-eq by fastforce
```

hence $\forall \mathcal{P} \in$ *probabilities*.
$(\sum b \leftarrow bids'.\ \mathcal{P}\ (bet\ b)) + total\text{-}price\ asks' + \varepsilon$
$\leq (\sum s \leftarrow asks'.\ \mathcal{P}\ (bet\ s)) + total\text{-}price\ bids' + k$
using *coherence-nonstrict-correspondence* $[of\ asks'\ bids'\ k - \varepsilon]$ **by** *auto*
thus *?rhs*
using ‹$0 < \varepsilon$› **by** *auto*
next
have $[bet\ b\ .\ b \leftarrow asks'^{\sim}\ @\ bids'] = \sim [bet\ s.\ s \leftarrow asks']\ @\ [bet\ b.\ b \leftarrow bids']$
(**is** - = $\sim$ *?ask-φs* @ *?bid-φs*)
unfolding *negate-bets-def*
by (*induct bids'*, *simp+*)
{
fix $\mathcal{P}$ :: $'a \Rightarrow real$
have $(\sum b \leftarrow asks'.\ \mathcal{P}\ (bet\ b)) = (\sum \varphi \leftarrow ?ask\text{-}\varphi s.\ \mathcal{P}\ \varphi)$
$(\sum b \leftarrow bids'.\ \mathcal{P}\ (bet\ b)) = (\sum \varphi \leftarrow ?bid\text{-}\varphi s.\ \mathcal{P}\ \varphi)$
by (*induct asks'*, *auto*, *induct bids'*, *auto*)
}
note $\star$ = *this*
let *?tot-bs* = *total-price bids'* **and** *?tot-ss* = *total-price asks'*
let *?c* = *?tot-ss* − *?tot-bs* − *k*
assume *?rhs*
have $\forall\ \mathcal{P} \in probabilities.\ (\sum b \leftarrow bids'.\ \mathcal{P}\ (bet\ b)) + ?c < (\sum s \leftarrow asks'.\ \mathcal{P}\ (bet\ s))$
using ‹*?rhs*› **by** *fastforce*
hence $\forall\ \mathcal{P} \in probabilities.\ (\sum \varphi \leftarrow ?bid\text{-}\varphi s.\ \mathcal{P}\ \varphi) + ?c < (\sum \varphi \leftarrow ?ask\text{-}\varphi s.\ \mathcal{P}\ \varphi)$
using $\star$ **by** *auto*
hence $\forall\ \mathcal{P} \in dirac\text{-}measures.\ (\sum \varphi \leftarrow ?bid\text{-}\varphi s.\ \mathcal{P}\ \varphi) + (\lfloor ?c \rfloor + 1) \leq (\sum \varphi \leftarrow ?ask\text{-}\varphi s.\ \mathcal{P}\ \varphi)$
using *strict-dirac-collapse* [*of ?bid-φs ?c ?ask-φs*]
by *auto*
hence *MaxSAT* ($\sim$ *?ask-φs* @ *?bid-φs*) + ($\lfloor ?c \rfloor$ + *1*) $\leq$ *length ?ask-φs*
by (*metis floor-add-int floor-mono floor-of-nat dirac-inequality-equiv*)
hence *MaxSAT* ($\sim$ *?ask-φs* @ *?bid-φs*) + *?c* < *length ?ask-φs*
by *linarith*
from *this* **obtain** ε :: *real* **where**
$0 < \varepsilon$
MaxSAT ($\sim$ *?ask-φs* @ *?bid-φs*) + *?c* + $\varepsilon \leq$ *length asks'*
using *less-diff-eq* **by** *fastforce*
hence π_{max} ⦇*asks* = *asks'*, *bids* = *bids'*⦈ $\leq k - \varepsilon$
using ‹$[bet\ b\ .\ b \leftarrow asks'^{\sim}\ @\ bids'] = \sim$ *?ask-φs* @ *?bid-φs*›
coherence-maxsat $[of\ asks'\ bids'\ k - \varepsilon]$
by *auto*
thus *?lhs* **using** ‹$0 < \varepsilon$› **by** *linarith*
qed

Below is our central result regarding coherence testing:

A strategy is incoherent if and only if there is a corresponding identity in probability theory that reflects it.

theorem (**in** *consistent-classical-logic*) *coherence-correspondence*:
(π_{max} ⦇ *asks* = *asks'*, *bids* = *bids'* ⦈ < *0*)

$= (\forall\ \mathcal{P} \in \textit{probabilities}.$
$(\sum b{\leftarrow}\textit{bids}'.\ \mathcal{P}\ (\textit{bet}\ b)) + \textit{total-price asks}'$
$< (\sum s{\leftarrow}\textit{asks}'.\ \mathcal{P}\ (\textit{bet}\ s)) + \textit{total-price bids}')$
using *coherence-strict-correspondence* **by** *force*

no-notation *Probability-Inequality-Completeness.relative-maximals* (‹$\mathcal{M}$›)

end

References

[1] B. De Finetti. Sui passaggi al limite nel calcolo delle probabilità. 63:1–12.

[2] A. Hájek. Scotching Dutch Books? 19:139–151.

[3] J. G. Kemeny. Fair bets and inductive probabilities. 20(3):263–273.

[4] R. S. Lehman. On Confirmation and Rational Betting. 20(3):251–262.

[5] F. P. Ramsey. Chapter 3. Truth and Probability. In J. B. Braithwaite, editor, *The Foundations of Mathematics and Other Logical Essays*, number 5 in The International Library of Philosophy $ Philosophy of Logic and Mathematics 32, Philosophy of Logic and Mathematics : In 8 Volumes. Routledge.

[6] H. R. Varian. The Arbitrage Principle in Financial Economics. 1(2):55–72.