

Frank–Wolfe Market Making

A tutorial walkthrough of

*Arbitrage-Free Combinatorial Market Making
via Integer Programming*

Christian Kroer · Miroslav Dudík · Sébastien Lahaie · Sivaraman Balakrishnan

ACM EC 2016 · arXiv:1606.02825v2, 10 June 2016

The paper is restated in its own terms, its derivations are filled in, and everything that is not the paper is labeled as such: implementation recommendations, research proposals, cautions, and field evidence. One numerical example, a six-security triangle market with a runnable program, runs through the whole text. Each chapter also compiles on its own.

Prepared from the discussion notes of 12 October 2026

Version 0.1 (draft)

How to read this document

This tutorial develops the market mechanism in “Arbitrage-Free Combinatorial Market Making via Integer Programming” by Kroer, Dudík, Lahaie, and Balakrishnan [P1]. It assumes convex optimization, integer programming, and probability. Begin with trade charges, derive the correction objective and its profit certificate, then implement the integer oracle on a market small enough to enumerate.

Reading paths

Goal	Read
Understand the contribution	Chapter 1 , then Chapters 6, 9 and 10
Derive the mathematics	Parts I and II , then Chapters 9, 10 and 12
Connect to operations research	Chapters 8, 9 and 11
Audit the small example	Chapters 15 and 16 , with Appendix A beside the code
Assess empirical evidence	Chapters 17 and 18 in the full edition
Plan a research prototype	Chapters 19 to 22 in the full edition

Notation and sources

The notation crosswalk defines the tutorial’s terms and their counterparts in the paper. A source result keeps its original number; a result numbered by this tutorial states the assumptions used here. Optional source-concordance sections preserve source-specific arguments and qualifications. Margin anchors refer to the arXiv v2 manuscript (arXiv:1606.02825v2, 10 June 2016), counting its title page as page 1. [Appendix D](#) maps chapters to source notes and paper locations; the notes are included in the repository.

The reading key distinguishes source restatements, tutorial derivations, implementation recommendations, and proposals. Evidence status stays beside the result it qualifies. Standalone chapters add short prerequisite reminders and their own bibliography; combined builds suppress those reminders. The revised Chapters 1–21 end with exercises on their central calculations or proposed implementation contracts.

Experiments and evidence

The notes dated 12 October 2026 report a completed six-security triangle experiment. Its numerical results are historical; the original raw JSON was not preserved. The original Python snapshot remains available with its full listing in [Appendix C](#). The revision manifest records artifact hashes and separates any new execution from that historical report.

The independent arithmetic and certificate checks supplied with this revision are new checks of the examples, with their own script and output. They do not recreate the historical execution environment or replicate the NCAA replay. The larger experiments in [Part VII](#) remain proposals. Public sources in [Part VI](#) retain their dated, attributed status; editing this tutorial does not establish current deployment or independently verified trading performance.

The full edition includes all 22 chapters. The core edition includes Chapters 1–16 and the appendices; references to omitted material link to the companion full PDF. Standalone PDFs also use that companion for cross-chapter links. Keep the PDFs and executable artifacts together as described in the repository README.

Contents

How to read this document	i
Reading key	ix
Notation	x
I The market	1
1 What the paper contributes	2
1.1 The problem	2
1.2 The integer solver supplies outcome columns	3
1.3 Prices, updates, and outside accounts	3
1.4 A first validation problem	3
2 Trades, cost functions, and prices	5
2.1 A trade has an immediate payment and a later payoff	5
2.2 The current price is a derivative	6
2.3 Why require a convex cost function?	6
2.4 Why choose the logarithmic cost function?	7
2.5 The categorical LMSR and the paper's compositional cost	7
2.6 Several binary markets can still quote inconsistent prices	8
3 Coherent prices and arbitrage	10
3.1 Start with complete, valid payoff vectors	10
3.2 Candidate prices are averages of valid payoffs	11
3.3 Coherent current prices rule out guaranteed profit	11
3.4 Incoherent current prices provide a profitable direction	12
3.5 Consistency identifies the constraint, but not the correction	12
II The projection	13
4 The convex conjugate	14
4.1 First ask what a trader with beliefs μ would earn	14
4.2 Changing variables produces the convex conjugate	14
4.3 Recovering the state for interior binary LMSR	15
4.4 Calculate the binary LMSR conjugate directly	15

4.5	Geometric interpretation: an affine lower bound	16
5	The correction objective	18
5.1	The cost function determines the correction objective	18
5.2	Derive the logarithmic formula by substitution	18
5.3	The direction follows from whose expectations are being taken	19
5.4	What the objective prefers	20
5.5	Why squared distance gives a different answer	20
5.5.1	Optional: local quadratic approximation	21
5.6	Liquidity scaling and constants	21
5.7	Optional: a market whose objective is squared distance	22
6	Bregman projection and guaranteed profit	23
6.1	Guaranteed profit is a different optimization	23
6.2	Every coherent candidate provides an upper bound	23
6.3	The projection problem	24
6.4	For interior binary LMSR, the smallest upper bound is attained	24
6.4.1	General value identity and finite attainment	25
6.5	Work through the identical-contract example	25
6.6	Coherence does not imply optimal profit extraction	26
6.7	Optional source concordance: Proposition 2.4	26
7	The loss budget	28
7.1	Define the loss before bounding it	28
7.2	Derive the loss budget	28
7.3	Compute the budget for LMSR	29
7.3.1	A budget and a current cash balance	30
7.3.2	Categorical blocks and nonuniform initialization	30
7.3.3	Changing the cost normalization changes no charges	30
III	The algorithm	32
8	Outcome spaces as integer programs	33
8.1	What must be exact	33
8.2	Standard logical constraints	34
8.3	The six-security triangle example	34
8.4	A tournament formulation you can build incrementally	35
8.4.1	Two encodings, different relaxations	36
8.5	Sums and comparisons	36
8.5.1	Heuristic initial prices	36
8.6	LCMM as an outer approximation	37
8.7	Optional source concordance	37
9	Frank–Wolfe as column generation	41
9.1	State the problem and its assumptions	41
9.2	The nonlinear master problem	41
9.3	The integer pricing oracle	42

9.4	The Frank–Wolfe gap	43
9.5	The basic column-generation loop	43
9.6	Categorical markets and the entropy extension	44
9.7	The exact correspondence to column generation	44
9.8	Scope of the smooth convergence rate	45
9.9	Sparsity, uniqueness, and what the mixture means	45
9.10	Optional comparison with a full joint LMSR	45
9.11	Optional source algorithm concordance	46
10	Safe updates and profit certificates	49
10.1	Converting a candidate price vector back to a state	49
10.2	Why the gap determines safe profit	49
10.3	Why a feasible price reset can still be wrong	50
10.4	The paper’s stopping conditions	51
10.5	Do not drop the constant from the stopping rule	52
10.6	Optional source proof concordance	52
11	Certificates from time-limited solves	54
11.1	Separate the incumbent from the objective bound	54
11.2	A useful action table	56
11.3	Absolute economic tolerances	56
11.4	A safe online integration protocol	57
12	Entropy boundaries, contraction, and the interior point	58
12.1	Why finite states cannot quote binary endpoints	58
12.2	Why ordinary smooth Frank–Wolfe analysis is insufficient	58
12.3	A coordinate-interior anchor	59
12.4	Keep derivatives finite without changing the certificate	59
12.4.1	Original-domain versus contracted-domain gap	60
12.5	Optional source concordance	61
IV	Running the mechanism	63
13	Partial outcomes and settlement	64
13.1	Partial outcomes and logical settlement	64
13.2	Settlement changes the cost function	64
13.2.1	What blockwise conditioning means	65
13.3	A compatible sequence preserves the loss budget	65
13.4	Optional source proof concordance	66
14	The mechanism end to end and the three ledgers	68
14.1	The Frank–Wolfe market maker, end to end	68
14.2	Separate three ledgers	69
14.2.1	Why safe internal trades preserve the bound	70
14.2.2	A real trade followed by an internal correction	71
14.3	A coherent state is not an always-coherent transaction rule	71

V	A worked example	72
15	The triangle market: an analytic projection	73
15.1	The analytic projection	73
15.2	Observed default results	75
15.3	Numerical and algorithmic scope	76
16	Running and auditing the code	77
16.1	What a successful run demonstrates	77
16.2	Environment and commands	78
16.3	Function contracts and evidence	78
16.4	Header and integer payoff model	78
16.5	The integer oracle	80
16.6	Cost, convex conjugate, state, and correction objective	81
16.7	The fully corrective step	82
16.8	The Frank–Wolfe loop	83
16.9	Numerical hull membership over enumerated outcomes	85
16.10	The ledger test	85
16.11	The main program	87
16.12	Reading the JSON output	90
16.12.1	Historical output schema	90
16.13	Versioned hardening and revision checks	93
VI	Evidence	94
17	The original NCAA experiments	95
17.1	Data, reconstruction, and engine settings	96
17.2	How orders were reconstructed	96
17.3	Initialization and settlement are reconstructed too	97
17.4	Forecast scores and reported results	97
17.5	What to conclude about scale	98
17.6	Code and data availability	98
18	Public evidence of deployment	99
18.1	What the public evidence establishes	99
18.1.1	Context and finding	99
18.1.2	How to interpret the evidence	100
18.1.3	Evidence by organization or project	101
18.2	Evidence about professional market makers	101
18.2.1	White Swan: the actual commercial problem	101
18.2.2	Susquehanna / SIG: venue-confirmed participation	102
18.2.3	Wintermute: reported venues, undisclosed algorithm	102
18.2.4	DRW: intended strategies in an official posting	102
18.2.5	How to use these findings	102
18.3	Totalis, integer optimization, and exact payoff models	103
18.3.1	The disclosed application and its limits	103
18.3.2	One feasible-state model, different jobs	103

18.3.3	Encode payoffs for every intended objective	104
18.4	Public implementations and their mathematical limits	104
18.4.1	OpenPolyTrader: the notes' reported code inspection	105
18.4.2	Why solver bounds matter	105
18.4.3	Polymarket's official maker bot: a quoting baseline	106
18.4.4	LayerX: explicitly paper trading	106
18.5	Arbitrage profits and the limits of deployment evidence	106
18.5.1	What the frequently repeated "\$40 million" result means	106
18.5.2	What would strengthen the deployment claim	107
VII	Building further	108
19	Implementation architecture	109
19.1	Build the research engine in layers	109
19.1.1	Existing fixtures and proposed capabilities	110
19.2	Define a precise oracle interface	111
19.2.1	Later optimization	112
19.3	Restricted-master choices	112
19.4	Implement order execution from the cost function	113
19.5	A minimal projection-job contract	114
19.6	Metrics and replay records	115
20	Validation experiments	117
20.1	Stage A: algebraic and economic validation	117
20.1.1	Fixtures and tolerance registry	117
20.1.2	Payoff and oracle contracts	118
20.1.3	Derivative, mixture and objective contracts	119
20.1.4	Profit, accounts and interruption contracts	119
20.2	Stage B: compare local constraints with the exact payoff hull	120
20.3	Stage C: small tournaments with enumeration	121
20.4	Stage D: partial settlement and logical propagation	122
20.4.1	Changing-cost accounts	122
20.5	Completion criteria for a first research prototype	123
21	Simulation, evaluation, and venue data	124
21.1	Stage E: a four-team pilot	124
21.1.1	Four treatments and their schedules	126
21.2	Measure four different outcomes	128
21.3	Replication and a common evaluation universe	129
21.4	Plots and action categories	130
21.5	Stage F: expand one factor at a time	130
21.6	Venue data requires an execution study	131
21.7	Implementation milestones	132

22 Research extensions	134
22.1 Stop oracle search when the economic certificate is sufficient	134
22.2 Turn oracle information into new LCMM cuts	134
22.3 Spend more time on cheap active-set updates	135
22.4 Design the security language to fit the algorithm	135
22.5 Separate information value from liquidity and compute	136
22.6 An order-book extension uses a different optimization problem	136
A Formula sheet and mistakes that change the method	138
A.1 Construct a candidate in one fixed problem	138
A.2 Certify the trade and interpret the return	139
A.3 Keep outside accounts separate	139
A.4 Mistakes that change the method	140
B Related papers and a reading plan	142
B.1 Orientation	142
B.2 The core path	142
B.2.1 A route through the pinned primary paper	143
B.3 Choose a follow-up by its implementation question	144
B.4 Recent adjacent work and the scope of its guarantees	145
B.5 Available implementation references	146
B.6 How to use the bibliography	146
C Complete listing of the example program	148
D Source concordance and artifacts	155
D.1 Paper locations	155
D.2 Source-note lookup	156
D.3 Executable and revision artifacts	156
References	157
Evidence sources	159
Index	161

Reading key

Paper identifies a restatement of the manuscript and cites its source. **Derivation** supplies a proof or example in tutorial notation, with its assumptions stated locally. **Implementation recommendation** and **Research proposal** describe suggested work. **Caution** marks a limitation; **Independent check** identifies a checked claim or source qualification. **Status** gives the provenance of a reported result. **Interpretation** distinguishes the notes’ assessment from a source’s own statement. Shaded boxes use the same vocabulary for longer passages.

Source restatements retain their original numbering and are explicitly identified as results of the paper. The main derivations use the tutorial’s notation. A qualification accompanies any source claim that needs narrower assumptions here; in particular, the generic inverse-state claim discussed in [Section 6.7](#) must not be used without its limitation. The paper’s $D(\mu||\theta)$ is the tutorial’s $F_\theta(\mu)$.

Evidence categories

A source can establish that a firm operates without revealing its algorithm, and a code repository can disclose an algorithm without demonstrating live operation. The evidence chapter keeps those questions separate.

Source	Supports	Does not by itself establish
Venue announcement or operator interview	Named activity and described workflow	Private algorithm or verified performance
First-party service description	An operational claim and disclosed method	Independent verification
Source code at an identified revision	Static implementation behavior	Live use or profitability
Paper trading or historical research	Reported experiment or retrospective measurement	Prospective execution
Observed transactions	Recorded behavior	The private program behind it

A job posting, README, or secondary report is identified by its actual source type. Claims retain their dates and the notes’ inspection limits. Historical triangle results, new revision checks, paper findings, and proposed experiments have separate provenance in the revision manifest.

Notation

One name is used for each object. The paper’s label and the identifier used in the example program are listed beside it so that the three can be read against each other. The paper writes the correction objective as the *mixed Bregman divergence* $D(\mu \parallel \theta)$ and calls the coherent price set the *marginal polytope*; those are source labels for the same objects.

Symbol	Name used here, and meaning	In the paper	In the program
Ω, ω	Outcome set and a complete outcome	Ω, ω	bits (x_1, x_2, x_3)
$\phi(\omega)$	Payoff map: the 0/1 payments of all securities in outcome ω	$\phi(\omega)$, payoff function	rows of <code>vertices</code>
z	Valid payoff vector, one element of $\mathcal{Z}$	z	<code>z</code>
$\mathcal{Z}$	Valid payoff set	$\mathcal{Z}$	<code>vertices</code> , <code>admitted</code>
$\mathcal{M}$	Coherent price set, $\text{conv}(\mathcal{Z})$	marginal polytope $\mathcal{M}$	<code>membership(mu,</code> <code>vertices)</code>
$\widetilde{\mathcal{M}}$	LCMM relaxation: an outer polyhedral approximation of $\mathcal{M}$	$\widetilde{\mathcal{M}}$, LCMM constraints	<code>A</code> , <code>rhs</code> (LP relaxation)
θ	Pricing state: the vector used to compute trade charges	market state θ	<code>theta</code>
θ^0	Initial pricing state	θ_0	<code>state(p0,</code> <code>b); zero in</code> <code>ledger_test</code> <code>delta</code>
δ	Trade: a signed bundle bought from the market maker	bundle δ	
C	Cost function; a trade is charged $C(\theta + \delta) - C(\theta)$	C	<code>cost(theta, b)</code>
b	Liquidity parameter of the LMSR	b	<code>b</code>
$p(\theta)$	Current price vector $\nabla C(\theta)$	instantaneous prices $p(\theta)$	<code>expit(theta /</code> <code>b)</code>
μ	Candidate price vector: proposed expected payoffs	coherent beliefs or prices μ	<code>mu</code>
R	Convex conjugate $\sup_s [s^\top \mu - C(s)]$	R	<code>entropy(mu, b)</code>
$\bar{R}$	Entropy formula, differentiated on active-coordinate interiors	$\bar{R}$	<code>entropy (via</code> <code>xlogy)</code>
$\hat{\theta}$	Entropy candidate state $\nabla \bar{R}(\mu)$ with prices μ	$\hat{\theta} = \nabla \bar{R}(\hat{\mu})$	<code>state(mu, b)</code>
$F_\theta(\mu)$	Correction objective $R(\mu) + C(\theta) - \theta^\top \mu$	$D(\mu \parallel \theta), F(\mu)$	<code>divergence(mu,</code> <code>theta, b)</code>
$\mu^\star$	The projection: minimizer of F_θ over $\mathcal{M}$	Bregman projection $\mu^\star$	<code>analytic,</code> <code>reference</code>
c	Oracle direction $\nabla \bar{R}(\mu) - \theta$; the candidate trade	$\nabla F(\mu_t) = \theta_t - \theta$	<code>c</code>
g	Exact Frank–Wolfe gap $c^\top \mu - \min_{z \in \mathcal{Z}} c^\top z$; upper-bounds objective suboptimality	FW gap $g(\mu)$	<code>g_exact (enumeration)</code>

Symbol	Name used here, and meaning	In the paper	In the program
$\bar{g}$	Upper gap from a valid oracle lower bound	— (added by the notes)	<code>g_upper</code>
m_{lb}	m_{lb} Oracle lower bound on $\min_z c^\top z$	—	<code>lower_bound</code>
m_{inc}	Feasible oracle objective; gives a lower gap estimate	—	<code>incumbent</code>
α	Extraction fraction; stop when $g \leq (1 - \alpha)F_\theta(\mu)$	approximation ratio α	<code>alpha</code>
ε	Contraction parameter	ε_t	<code>eps</code>
u	Interior anchor point	interior point u	<code>u</code>
$\mathcal{Z}_t$	Active vertex set of Frank–Wolfe	$\mathcal{Z}_t$	<code>active</code>
σ	Partial outcome: securities already settled	σ	— (not in the example)
$B(\theta^0)$	Loss budget: bound on market-maker loss from θ^0	$\max_\omega D(\phi(\omega) \parallel \theta_0)$	<code>loss_bound</code>
$L(z)$	Realized market-maker loss at outcome z	—	<code>loss</code>
q, v, K	Real trader holdings, cumulative virtual corrections, real cash	—	<code>q</code> , (implicit), <code>cash</code>

The correction objective F_θ is an optimized expected-profit value; L is a realized accounting quantity. Entropy gradients and inverse states use only active interior coordinates: $0 < \mu_i < 1$ in the reduced binary representation. Settled coordinates use the changed cost of [Chapter 13](#). The historical program retains its original identifiers.

Part I

The market

Chapter 1

What the paper contributes

The paper uses integer optimization to make related contract prices consistent without enumerating every outcome. Its update rule preserves the market maker’s original worst-case loss bound.

1.1 The problem

Suppose two contracts each pay one dollar when the same event occurs. If their current marginal prices are 0.8 and 0.4, a trader can sell a small quantity of the first and buy the same quantity of the second. The settlement payoffs cancel. For a sufficiently small trade, the sale brings in more than the purchase costs. Consistent prices would assign both contracts the same price.

With many related events, consistency is harder to enforce. Tournament contracts can concern a team’s game wins, its total wins, and the eventual champion. Their prices are *coherent prices* when they are expected contract payoffs under one probability distribution over valid tournament outcomes. A fixed 64-team single-elimination bracket has 63 games and 2^{63} possible complete outcomes, too many to enumerate in a variable-per-outcome formulation.

Kroer, Dudík, Lahaie, and Balakrishnan address this problem in *Arbitrage-Free Combinatorial Market Making via Integer Programming* [P1]. This tutorial follows the 21-page arXiv v2 manuscript dated June 10, 2016, for readers familiar with convex optimization, integer programming, and probability.

Paper. The framework starts with a bounded-loss cost function. The implementation studied in the paper uses a sum of categorical logarithmic market scoring rule (LMSR) costs. These costs quote prices within each block, but do not enforce all relationships between blocks. An integer model specifies which settlement payoff vectors are possible. An outer Frank–Wolfe algorithm then adjusts a mixture of those vectors to select a correction compatible with the base cost’s trade charges. Later chapters state the regularity and certificate conditions needed for that correction.  §§1–4

The paper also uses a faster linear-constraint method for partial corrections before invoking integer optimization. [Chapter 14](#) gives the full mechanism ([Mechanism 1](#)); [Chapters 9](#) and [12](#) give the projection and initialization algorithms ([Algorithms 2](#) and [3](#)). The tutorial develops the accounting first with reduced binary LMSR markets.

1.2 The integer solver supplies outcome columns

The integer model describes a *valid payoff vector*: the zero-or-one payoff of each traded contract in one possible outcome. The outer algorithm stores discovered payoff vectors and chooses nonnegative weights summing to one. Their weighted average is a coherent price vector.

To improve that mixture, Frank–Wolfe asks the integer solver to minimize a linear function of a payoff vector. The outer algorithm supplies the coefficients; the solver returns another feasible vector. That vector becomes a column of the restricted master problem, whose objective is convex and nonlinear. The logical payoff constraints stay fixed during a projection, while the linear objective coefficients change between iterations.

This is the connection with Dantzig–Wolfe decomposition and column generation. Here a *pricing oracle* finds an entering outcome column. Contract quotes come from the cost function. The integer solver therefore handles a linear objective; the outer algorithm handles the entropy-based correction objective.

1.3 Prices, updates, and outside accounts

Distinction	Consequence
Valid payoff vector and coherent price vector	The MIP supplies one possible settlement. The master forms expected payoffs from a mixture of settlements. Mixture weights sum to one; overlapping-event prices need not.
Coherent prices and a safe state update	Coherence describes the candidate prices. Preserving the loss bound requires a separate certificate for the hypothetical trade that changes the pricing state.
Pricing state and outside positions	Internal corrections change future quotes while leaving actual trader holdings and cash unchanged at that instant.
Accepted correction and unchanged return	On interruption, the method returns a certified coherent state when one is available; otherwise it keeps the old state, which can retain arbitrage.

An accepted coherent state removes arbitrage at its current prices even if the hypothetical correction profit is below the optimum. Subsequent real trades can create new inconsistencies. The mechanism’s loss guarantee does not supply a polynomial-time algorithm for arbitrary outcome models or a fast exact evaluation of a joint LMSR partition function [P1, §§2.3, 4, 5.2, 6].

1.4 A first validation problem

Start with three binary events and the three contracts paying on their pairwise conjunctions. This *triangle market* has eight underlying outcomes and six traded contracts. Enumeration is small enough to check the integer model and every settlement profit independently. The experiment should let you:

- Exhibit prices satisfying every chosen local constraint while violating a global constraint.
- Recover the coherent projection by enumeration and by an IP oracle.

- Verify the correction profit in every possible outcome.
- Demonstrate that an arbitrary coherent price reset can fail the profit certificate.
- Track actual trader liabilities separately from internal state changes.

Chapter 15 derives the analytic answer and Chapter 16 explains the starter implementation. This is the tutorial's validation example; the paper's historical experiment used NCAA tournament markets.

Exercises.

1. For the two identical contracts, write the payoff of selling h units of the first and buying h units of the second. Use the marginal prices to find the first-order trade charge as $h \downarrow 0$.
2. In the triangle market, suppose every event has probability $1/2$ and every pairwise conjunction has probability zero. Each pair has a valid probability table. Explain why one joint distribution cannot have these marginals. Hint: at most one event could occur in any outcome.

Chapter 2

Trades, cost functions, and prices

Derive the charge and settlement payoff of a signed trade, then obtain marginal prices for binary and categorical markets.

A contract pays one dollar if its event occurs and zero otherwise. Traders may buy or sell any real quantity, including a short position. The model has no fees, borrowing limits, or order-book execution. These assumptions describe the paper's cost-function market [P1, §2.1].

2.1 A trade has an immediate payment and a later payoff

Let $z \in \{0, 1\}$ be the contract's realized payoff. Let θ be the *pricing state*, and let δ be the number of shares a trader buys. A negative δ is a sale.

The market uses a function C , called the *cost function*, to charge for the trade:

$$\text{charge} = C(\theta + \delta) - C(\theta). \quad (2.1)$$

At settlement, the trader receives δz . The trader's profit is therefore

$$\delta z - C(\theta + \delta) + C(\theta).$$

A negative charge means that the market pays the trader immediately. A negative settlement payoff means that the trader owes money at settlement.

For example, use $C(\theta) = \log(1 + e^\theta)$ and start at $\theta = 0$. Selling one share means $\delta = -1$, with signed charge

$$C(-1) - C(0) = \log(1 + e^{-1}) - \log 2 \approx -0.3798855.$$

The trader receives 0.3798855 immediately. If the event fails, settlement pays zero and the profit is 0.3798855. If the event occurs, the trader owes one dollar and the profit is -0.6201145 . We derive this cost function's prices in [Section 2.4](#).

The level $C(\theta)$ is not the market's cash balance. Only a change in the cost function determines a trade's charge. Adding a constant to C changes no trade charge.

Paper. Let Ω be the finite set of possible final outcomes. A security i has payoff

■ §2.1

$$\phi_i : \Omega \longrightarrow \{0, 1\}.$$

For example, $\phi_i(\omega) = 1$ if a specified team is champion in outcome ω , and zero otherwise. Write $\phi(\omega) \in \{0, 1\}^d$ for the vector of all security payoffs, where d is the number of securities.

A trader buys a bundle $\delta \in \mathbb{R}^d$. Negative components represent sales or short positions. At settlement the bundle pays $\delta^\top \phi(\omega)$. The theoretical model allows these signed bundles; an implementation with position or borrowing restrictions has a different set of feasible trader actions [P1, §2.1].

2.2 The current price is a derivative

For a small signed trade δ , differentiability gives

$$C(\theta + \delta) - C(\theta) = C'(\theta)\delta + o(|\delta|).$$

The error divided by $|\delta|$ tends to zero as $\delta \rightarrow 0$. We therefore define the *current price* by $p = C'(\theta)$. For a finite trade, the exact charge is

$$C(\theta + \delta) - C(\theta) = \int_0^\delta C'(\theta + t) dt.$$

The current price describes the first small part of a trade. It generally cannot be multiplied by the entire quantity to recover the charge.

For several contracts, the pricing state and trade are vectors. The same rule gives the *current price vector*

$$p(\theta) = \nabla C(\theta) \tag{2.2}$$

(the paper's instantaneous price vector) and the charge $C(\theta + \delta) - C(\theta)$. The settlement payoff is the inner product $\delta^\top z$. For a scalar size s and a vector direction $a \in \mathbb{R}^d$,

$$C(\theta + sa) - C(\theta) = \int_0^s a^\top p(\theta + ta) dt.$$

Chapter 17 uses the distinction between marginal and average execution prices when reconstructing historical orders.

2.3 Why require a convex cost function?

Convexity implies that the price cannot decrease as a trader buys more of the same contract while holding the other coordinates fixed. More generally, it gives the supporting inequality

$$C(\theta + \delta) - C(\theta) \geq \delta^\top p.$$

Section 3.3 combines this supporting inequality with coherence to rule out a trade that profits in every valid outcome.

Convexity alone does not ensure that prices lie between zero and one. It also does not specify how much the market maker can lose. Those properties depend on the particular cost function. Convex cost functions and payoff constraints form the design framework developed by Abernethy, Chen, and Vaughan [P3].

2.4 Why choose the logarithmic cost function?

For a binary contract, the paper's *logarithmic market scoring rule*, abbreviated LMSR, reduces to

$$C(\theta) = b \log(1 + e^{\theta/b}), \quad b > 0.$$

Here b is the *liquidity parameter*. Differentiating gives

$$p = \frac{e^{\theta/b}}{1 + e^{\theta/b}}, \quad \frac{dp}{d\theta} = \frac{p(1-p)}{b}.$$

These formulas explain several useful properties of the choice:

- Every finite pricing state produces a price strictly between zero and one.
- Purchases increase the price smoothly.
- At the same initial price, a larger b reduces the price change caused by a fixed signed trade.
- Starting from $\theta = 0$, the worst-case loss budget is $b \log 2$ for one binary market. [Chapter 7](#) derives this bound.

Starting at $\theta = 0$, a purchase of one share moves the price from 0.5 to $1/(1 + e^{-1}) \approx 0.7311$ when $b = 1$, and to $1/(1 + e^{-1/2}) \approx 0.6225$ when $b = 2$. At a fixed nonzero state, changing b also changes the initial price; [Section 5.6](#) keeps these comparisons separate. LMSR also connects trade profit to changes in logarithmic forecast scores. That connection motivates the design, but the cash-and-payoff model is sufficient to derive the mathematics used here [\[P2\]](#).

LMSR is a design choice. After choosing its trade charges, we derive the corresponding correction objective in [Chapters 4 to 6](#).

2.5 The categorical LMSR and the paper's compositional cost

For mutually exclusive and exhaustive outcomes $x \in \mathcal{X}$, the LMSR cost with liquidity parameter $b > 0$ is

$$C(\theta) = b \log \sum_{x \in \mathcal{X}} \exp(\theta_x/b), \quad p_x(\theta) = \frac{\exp(\theta_x/b)}{\sum_y \exp(\theta_y/b)}. \quad (2.3)$$

At fixed initial prices, increasing b reduces the change in the marginal bundle quote $a^\top p$ caused by a given trade sa , and increases the loss budget [\[P1, Eq. \(4\); P2\]](#).

Adding the same constant k to every component of a categorical block leaves its probabilities unchanged. For a complete categorical block, the realized payoff z is one-hot, so

$$C(\theta + k\mathbf{1}) = C(\theta) + k, \quad (k\mathbf{1})^\top z = k.$$

The extra cost and settlement payoff cancel in trade profit. This constant shift is a cash-equivalent component because one share of every category pays one dollar in every outcome. The argument uses a complete block; arbitrary collections of binary contracts need not have a fixed sum of payoffs.

Paper. There are random variables X_j with finite domains $\mathcal{X}_j$. For each variable, the [§3.1, Eq. \(4\)](#)

securities are the indicators $\phi_{j,x} = \mathbf{1}\{X_j = x\}$. The cost is

$$C(\theta) = \sum_{j=1}^m b \log \sum_{x \in \mathcal{X}_j} \exp(\theta_{j,x}/b).$$

Each categorical market is internally normalized, but the blocks do not automatically respect cross-variable logic. For example, the championship probability must equal the probability of winning six games for a team in a fixed six-round bracket. Those two contracts have identical payoffs, yet separately normalized blocks can quote different prices for them [P1, §§2.1, 3.1].

Implementation recommendation. Start with a common b , as in the paper. Separate b_j values are a straightforward mathematical extension, replacing b by b_j in each block, but they change the relative weights in the projection and the total loss budget. Treat that as an explicit market-design choice.

2.6 Several binary markets can still quote inconsistent prices

For a binary categorical market with Yes and No states q_Y, q_N , factor out the No-state exponential:

$$C(q_Y, q_N) = b \log(e^{q_Y/b} + e^{q_N/b}) = q_N + b \log(1 + e^{(q_Y - q_N)/b}).$$

Write $\theta_i = q_Y - q_N$. The component q_N contributes equally to cost and settlement payoff, since exactly one category pays. Storing the relative state therefore gives the reduced cost and Yes price

$$C_i(\theta_i) = b \log(1 + \exp(\theta_i/b)), \quad p_i(\theta_i) = \frac{1}{1 + \exp(-\theta_i/b)}. \quad (2.4)$$

This is a reduction of two-outcome LMSR, not a different pricing rule.

For d separately priced binary contracts, sum these costs:

$$C(\theta) = b \sum_{i=1}^d \log(1 + e^{\theta_i/b}).$$

This is the representation used in the executable example of [Chapter 16](#). Each coordinate has the binary price formula above. Adding a trade to one coordinate does not change another coordinate's price. This computational convenience leaves relationships between contracts unenforced.

Our running example has two contracts that pay on the *same event*. The only valid payoff vectors are $(0, 0)$ and $(1, 1)$. Suppose the current prices are

$$p = (0.8, 0.4), \quad b = 1.$$

The corresponding pricing state follows by solving the price equation for θ_i :

$$\theta_i = b \log \frac{p_i}{1 - p_i}, \quad \theta = (\log 4, \log(2/3)).$$

The two prices disagree even though the payoffs are identical. Take the trade $\delta = t(-1, 1)$ for $t > 0$: sell the first contract and buy the second. Its settlement payoff is zero, and its

first-order charge is $t(-0.8 + 0.4) = -0.4t$. Differentiability makes the exact charge negative for sufficiently small t , so the immediate receipt is a guaranteed profit. [Section 6.5](#) calculates the best such trade exactly.

The reduced binary formula still represents two possible payoffs. [Section 4.4](#) therefore derives contributions from both Yes and No in its convex conjugate.

Exercises.

1. At $\theta = 0$ and $b = 1$, buy two binary shares. Calculate the charge, the average price per share, and the profit for each settlement outcome. Compare the average price with the initial marginal price 0.5.
2. In a complete categorical block, buy k shares of every category. Verify that the charge and settlement payoff are both k .
3. Repeat the one-share purchase at $b = 2$. Compare the price movement and the initial loss budgets derived in [Section 7.3](#).

Chapter 3

Coherent prices and arbitrage

Characterize coherent prices as expected valid payoffs and prove when a signed trade can earn a strictly positive minimum profit.

There are finitely many possible outcomes and d binary contracts. Signed trades of any real size are allowed. The cost function C is finite, convex, and differentiable on $\mathbb{R}^d$. At the current pricing state θ , prices are $p = \nabla C(\theta)$.

3.1 Start with complete, valid payoff vectors

Let $\mathcal{Z} \subseteq \{0, 1\}^d$ be the nonempty *valid payoff set*. A vector $z \in \mathcal{Z}$ lists what every contract pays in one possible outcome.

Logical relationships belong in $\mathcal{Z}$. For example, if the third contract pays when both of the first two events occur, then every valid payoff satisfies

$$z_3 = z_1 z_2.$$

For a probability distribution on the valid payoff set, let $\mu_i = \mathbb{E}[z_i]$ be the expected payoff of contract i . These expectations will define coherent prices. The product equation describes realized zero-or-one payoffs; it does not imply that their expectations satisfy $\mu_3 = \mu_1 \mu_2$. The latter equation would impose an independence assumption on the two events. In general,

$$\mu_3 = \mathbb{E}[z_1 z_2]$$

can differ from $\mathbb{E}[z_1] \mathbb{E}[z_2]$.

The four possible payoff vectors, with one example distribution, are

z_1	z_2	$z_3 = z_1 z_2$	Probability
0	0	0	1/2
1	0	0	0
0	1	0	0
1	1	1	1/2

Here $\mu = (1/2, 1/2, 1/2)$: the conjunction has expectation 1/2, while $\mu_1 \mu_2 = 1/4$. Coherence allows dependence between the events.

3.2 Candidate prices are averages of valid payoffs

A candidate price vector μ is coherent if some probability distribution over $\mathcal{Z}$ has expectation μ . Explicitly, there must be weights λ_z satisfying

$$\mu = \sum_{z \in \mathcal{Z}} \lambda_z z, \quad \lambda_z \geq 0, \quad \sum_{z \in \mathcal{Z}} \lambda_z = 1.$$

The weights λ_z sum to one. The entries of μ need not, because several contracts can pay together; the preceding example sums to $3/2$. Let Ω be the finite set of complete outcomes, and let $\phi : \Omega \rightarrow \{0, 1\}^d$ list their traded contract payoffs. The *coherent price set* is the convex hull of the valid payoff set:

$$\mathcal{Z} = \{\phi(\omega) : \omega \in \Omega\}, \quad \mathcal{M} = \text{conv}(\mathcal{Z}). \quad (3.1)$$

The symbol conv denotes all such convex combinations. The paper calls $\mathcal{M}$ the marginal polytope. A coherent vector can have several distribution witnesses [P1, §2.2; P3].

For the two identical contracts in Section 2.6,

$$\mathcal{Z} = \{(0, 0), (1, 1)\}, \quad \mathcal{M} = \{(r, r) : 0 \leq r \leq 1\}.$$

Every coherent candidate gives both contracts the same price. The current prices $(0.8, 0.4)$ lie outside this line segment.

3.3 Coherent current prices rule out guaranteed profit

Here an *arbitrage* means a trade whose minimum profit over all valid outcomes is strictly positive. This is the guaranteed-profit criterion used in the paper. The propositions below use the finite convex differentiable cost and unrestricted signed trades stated above; they do not assume LMSR.

Proposition 3.1. *If the current price vector p belongs to $\mathcal{M}$, no trade guarantees a strictly positive profit in every valid outcome.*

Derivation. Consider a trade $\delta \in \mathbb{R}^d$. The convex supporting inequality from Section 2.3 gives

$$C(\theta + \delta) - C(\theta) \geq \delta^\top p.$$

Since p is an average of valid payoffs, the smallest settlement payoff cannot exceed its average:

$$\min_{z \in \mathcal{Z}} \delta^\top z \leq \delta^\top p.$$

Subtract the charge from both sides and combine the two inequalities. The trade's guaranteed profit satisfies

$$\min_{z \in \mathcal{Z}} [\delta^\top z - C(\theta + \delta) + C(\theta)] \leq 0.$$

Thus the trade cannot profit in every valid outcome [P1, §2.2; P3].

The argument has a direct interpretation. The market charges at least the payoff's value at the current prices. One possible outcome pays no more than that value. The trader therefore cannot make a strictly positive profit in every outcome.

3.4 Incoherent current prices provide a profitable direction

Proposition 3.2. *Under the stated assumptions, if $p \notin \mathcal{M}$, some sufficiently small trade guarantees a strictly positive profit.*

Derivation. A point outside a compact convex set can be strictly separated from that set by a linear function. Choose the separating vector's sign so that every valid payoff has larger value than the current prices. There are then a vector a and a margin $\eta > 0$ such that

$$\min_{z \in \mathcal{Z}} a^\top z - a^\top p = \eta.$$

Consider the trade $\delta = ta$, where $t > 0$. Differentiability gives

$$C(\theta + ta) - C(\theta) = ta^\top p + o(t).$$

Write the remainder as $r(t)$, with $r(t)/t \rightarrow 0$ as $t \downarrow 0$. Choose $t > 0$ small enough that $|r(t)| < t\eta/2$. The trade's guaranteed profit is then $t\eta - r(t) > t\eta/2 > 0$ [P1, §2.2; P3].

In the identical-contract example, choose $a = (-1, 1)$. Every valid payoff has $a^\top z = 0$, while $a^\top p = -0.4$. A small sale of the expensive contract and purchase of the cheap contract therefore yields a positive immediate receipt and no settlement liability.

3.5 Consistency identifies the constraint, but not the correction

There are infinitely many coherent candidates (r, r) . Coherence alone does not tell us whether to choose $r = 0.5$, 0.6 , or another value.

The cost function supplies the missing criterion. Chapter 4 calculates the greatest expected profit available under a candidate price vector. Chapter 6 then proves why minimizing that quantity over $\mathcal{M}$ solves the guaranteed-profit problem.

The guarantee covers the outcomes in $\mathcal{Z}$. Omitting a real settlement possibility can invalidate it. Trading restrictions can also invalidate the converse: the separating trade might not be allowed.

Exercises.

1. Replace the joint table above by the uniform distribution on its four rows. Calculate μ and determine whether $\mu_3 = \mu_1\mu_2$.
2. For two identical contracts with $p = (0.8, 0.4)$, $b = 1$, and binary LMSR charges, calculate the exact charge of $0.1(-1, 1)$ and its profit in both valid outcomes. Compare with the first-order charge -0.04 .
3. If sales are forbidden, determine which step of the separation proof no longer guarantees an available trade.

Part II

The projection

Chapter 4

The convex conjugate

Derive the trader's best expected-profit value, calculate the binary LMSR conjugate, and interpret its affine lower bound.

4.1 First ask what a trader with beliefs μ would earn

Let C be finite, convex, and differentiable on $\mathbb{R}^d$. A signed trade $\delta \in \mathbb{R}^d$ changes the pricing state from θ to $\theta + \delta$. It is charged $C(\theta + \delta) - C(\theta)$ (the trade-charge rule in [Equation \(2.1\)](#)) and pays $\delta^\top z$ at settlement. Since $\mathbb{E}[z] = \mu$ for a trader with beliefs μ , that trader evaluates the trade by

$$\text{expected profit} = \delta^\top \mu - C(\theta + \delta) + C(\theta).$$

The trader chooses δ without budget or position limits. Denote the best expected-profit value by $F_\theta(\mu)$:

$$F_\theta(\mu) = \sup_{\delta \in \mathbb{R}^d} [\delta^\top \mu - C(\theta + \delta) + C(\theta)]. \quad (4.1)$$

We will use this value as the *correction objective*. [Chapter 6](#) explains why minimizing it over coherent prices solves the guaranteed-profit problem. The subscript records the current pricing state, which is fixed during this optimization. The supremum is the least upper bound of achievable expected profits. For the binary LMSR, a finite trade attains it when every candidate price μ_i lies strictly between zero and one. Boundary candidate prices may require an unbounded sequence of trades.

This definition already gives $F_\theta(\mu) \geq 0$, because the trader can choose $\delta = 0$. It also identifies the units: dollars, under the convention that each winning share pays one dollar.

4.2 Changing variables produces the convex conjugate

Let $s = \theta + \delta$ be the trader's chosen new pricing state. Then $\delta = s - \theta$, so

$$\begin{aligned} F_\theta(\mu) &= \sup_s [(s - \theta)^\top \mu - C(s) + C(\theta)] \\ &= \sup_s [s^\top \mu - C(s)] + C(\theta) - \theta^\top \mu. \end{aligned}$$

Only the bracketed expression depends on the new pricing state. Its optimized value has a standard definition:

$$R(\mu) = \sup_{s \in \mathbb{R}^d} [s^\top \mu - C(s)]. \quad (4.2)$$

The function R is the *convex conjugate* of C , written $R = C^*$. Substituting the definition gives

$$F_\theta(\mu) = R(\mu) + C(\theta) - \theta^\top \mu. \quad (4.3)$$

Paper. The convex conjugate has removed the optimization over trades and expressed its result as a function of candidate prices. This is the economic role of the conjugate in the paper, which introduces R by Eq. (1) of the paper and then defines the *mixed Bregman divergence* $D(\mu \parallel \theta) = R(\mu) + C(\theta) - \theta \cdot \mu$, the object written $F_\theta(\mu)$ here [P1, §2.2].

Once C is fixed, its defining supremum determines R .

4.3 Recovering the state for interior binary LMSR

For a finite differentiable convex cost, the general conjugacy relation is

$$\mu = \nabla C(\hat{\theta}) \iff \hat{\theta} \in \partial R(\mu).$$

Here $\partial R(\mu)$ is the set of supporting slopes: a vector s belongs to it when $R(v) \geq R(\mu) + s^\top (v - \mu)$ for every v . A finite state in this set attains the conjugate supremum and quotes μ . The set can contain several states or be empty.

For binary LMSR at $0 < \mu_i < 1$, the maximizing state is unique and

$$\hat{\theta} = \nabla R(\mu).$$

The two gradients convert between quantities: ∇C converts a pricing state into prices, and ∇R converts interior candidate prices into a pricing state. The functions C and R themselves are not inverse functions [B1, §3.3].

The inverse-gradient formula uses the stated interior assumptions. The counterexample in Section 6.7 shows why an arbitrary inverse-price state for a nonsmooth conjugate need not extract optimal guaranteed profit.

4.4 Calculate the binary LMSR conjugate directly

For one binary contract, take

$$C(s) = b \log(1 + e^{s/b}), \quad b > 0.$$

For a candidate price $0 < \mu < 1$, maximize

$$s\mu - b \log(1 + e^{s/b}).$$

The derivative vanishes when

$$\mu = \frac{e^{s/b}}{1 + e^{s/b}}, \quad s = b \log \frac{\mu}{1 - \mu}.$$

At this state, $1 + e^{s/b} = 1/(1 - \mu)$. Substitution gives

$$\begin{aligned} R(\mu) &= b\mu \log \frac{\mu}{1 - \mu} + b \log(1 - \mu) \\ &= b [\mu \log \mu + (1 - \mu) \log(1 - \mu)]. \end{aligned}$$

The full definition includes its domain:

$$R(\mu) = \begin{cases} b[\mu \log \mu + (1 - \mu) \log(1 - \mu)], & 0 \leq \mu \leq 1, \\ +\infty, & \text{otherwise,} \end{cases} \quad (4.4)$$

with $0 \log 0 = 0$ by continuity. The endpoints also follow directly from the supremum: for $\mu = 0$, $-C(s) \rightarrow 0$ as $s \rightarrow -\infty$; for $\mu = 1$, $s - C(s) \rightarrow 0$ as $s \rightarrow +\infty$. Both expressions are strictly below zero at every finite state, so neither endpoint supremum is finitely attained. For $\mu > 1$, sending $s \rightarrow +\infty$ makes $s\mu - C(s)$ diverge. For $\mu < 0$, sending $s \rightarrow -\infty$ does the same. This explains why the finite domain is exactly the interval of possible binary expectations.

For separate binary markets, the maximization separates across coordinates. The convex conjugate is the sum of these one-coordinate formulas, finite on the whole cube $[0, 1]^d$. Coherence imposes the additional restriction $\mu \in \mathcal{M}$, which may be a strict subset of that cube. Both the Yes and No terms are required even when only the Yes price is stored.

The formula is negative on the interior and equals zero at either endpoint. The conjugate need not be nonnegative. The nonnegative expected-profit value is $F_\theta(\mu) = R(\mu) + C(\theta) - \theta^\top \mu$, which includes both the cost constant and the belief-dependent linear term.

For a numerical calculation, take $b = 1$, $\theta = 0$, and $\mu = 3/4$. The maximizing state is $s = \log 3$, and

$$R(3/4) = \frac{3}{4} \log 3 - \log 4 = \frac{3}{4} \log \frac{3}{4} + \frac{1}{4} \log \frac{1}{4} \approx -0.562335.$$

Adding $C(0) = \log 2$ gives $F_0(3/4) \approx 0.130812$.

Paper. In the paper's unit-liquidity categorical LMSR example, $R(\mu)$ equals negative entropy whenever μ is a probability distribution and $+\infty$ otherwise, and the mixed Bregman divergence is the Kullback–Leibler divergence [P1, Ex. 2.3]. With general liquidity $b > 0$, both the negative-entropy expression and the KL divergence are multiplied by b . Chapter 5 derives that logarithmic form.

■ Ex. 2.3

4.5 Geometric interpretation: an affine lower bound

For a candidate price vector with $R(\mu) < \infty$, the defining supremum implies, for every state s ,

$$s^\top \mu - R(\mu) \leq C(s).$$

For fixed μ , the left side is an affine function of the state. In one dimension it is a line with slope μ . The value $R(\mu)$ sets the line's intercept so that the line is as high as possible while remaining below the cost function.

If the supremum is attained at a finite state $\hat{\theta}$, the affine lower bound touches the cost function there and is a supporting line. At an endpoint belief, the lower bound can instead be approached asymptotically. At the current state θ , the vertical difference is

$$C(\theta) - [\theta^\top \mu - R(\mu)] = F_\theta(\mu).$$

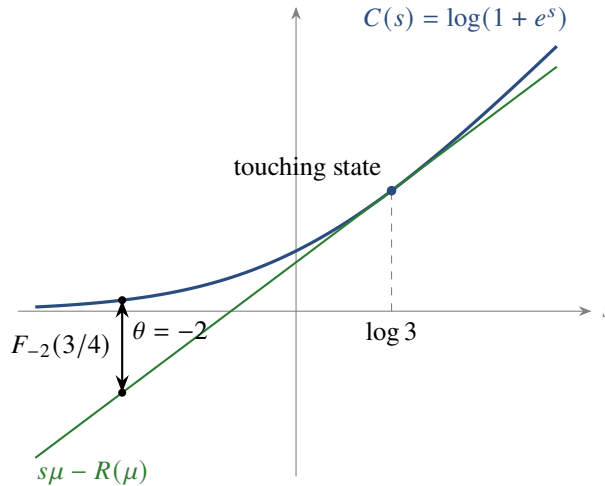


Figure 4.1: Binary LMSR with $b = 1$ and candidate $\mu = 3/4$. The supporting line has slope $3/4$ and intercept $-R(3/4) \approx 0.562335$; it touches the cost at $\theta = \log 3$. At the illustrated current state $\theta = -2$, the vertical gap is the best expected-profit value $F_{-2}(3/4)$.

The same quantity therefore has two interpretations: it is the vertical difference in Figure 4.1, and it is the trader's best expected-profit value under μ [B1, §3.3].

Finally, replacing C by $C + k$ replaces R by $R - k$. Their changes cancel in F_θ . Choosing a convenient zero for the cost function changes none of the economics.

Exercises.

1. With $b = 1$, $\theta = 0$, and $\mu = 3/4$, calculate the optimal trade, its charge, and its expected settlement payoff. Check that their difference equals $F_0(3/4)$ above.
2. For $\mu = 0$, identify the affine lower bound and prove that it never touches the binary cost at a finite state.

Chapter 5

The correction objective

Derive the binary logarithmic objective and compare its coherent correction with squared distance and reversed KL.

5.1 The cost function determines the correction objective

The expected-profit derivation in [Section 4.2](#) gives

$$F_{\theta}(\mu) = R(\mu) + C(\theta) - \theta^{\top} \mu,$$

where $R = C^*$ is the convex conjugate. This chapter substitutes the binary LMSR formula into the correction objective. The correction problem is

$$\min_{\mu \in \mathcal{M}} F_{\theta}(\mu).$$

The constraint requires all candidate prices to be consistent with one distribution over valid payoffs. The objective selects among those candidates. [Chapter 6](#) proves that the resulting minimum equals the best guaranteed-profit value.

For this guaranteed-profit problem, the trade-charge rule determines F_{θ} . A different discrepancy function solves a different optimization problem.

5.2 Derive the logarithmic formula by substitution

For this cost function, the convex conjugate is ([Equation \(4.4\)](#))

$$R(\mu) = b \sum_i [\mu_i \log \mu_i + (1 - \mu_i) \log(1 - \mu_i)].$$

The current prices give two useful identities:

$$\theta_i = b \log \frac{p_i}{1 - p_i}, \quad C_i(\theta_i) = -b \log(1 - p_i),$$

where C_i denotes the one-coordinate term in C . For one coordinate, the substitution is

$$\begin{aligned} F_{\theta,i}(\mu_i) &= b\mu_i \log \mu_i + b(1 - \mu_i) \log(1 - \mu_i) \\ &\quad - b\mu_i \log \frac{p_i}{1 - p_i} - b \log(1 - p_i). \end{aligned}$$

The last two terms contribute $-b\mu_i \log p_i$ and $-b(1 - \mu_i) \log(1 - p_i)$. Combining each with its entropy term and summing over coordinates gives

$$F_\theta(\mu) = b \sum_i \left[\mu_i \log \frac{\mu_i}{p_i} + (1 - \mu_i) \log \frac{1 - \mu_i}{1 - p_i} \right]. \quad (5.1)$$

Thus F_θ is b times the sum of Bernoulli Kullback–Leibler (KL) divergences, with the candidate probabilities in the first argument and current prices in the second. KL is not symmetric [P1, §2.2; B1, §3.3]. ■ §2.2

The formula applies at candidate endpoints by continuity, using $0 \log 0 = 0$. Its derivatives require interior candidate prices.

Derivation. For the paper’s compositional cost, a sum of categorical LMSR blocks each of the form (2.3), where j indexes blocks and $x \in \mathcal{X}_j$ their categories, the same substitution gives

$$F_\theta(\mu) = b \sum_j \sum_{x \in \mathcal{X}_j} \mu_{j,x} \log \frac{\mu_{j,x}}{p_{j,x}(\theta)}.$$

This is a sum of KL divergences between categorical marginals. In the reduced binary case it is the sum of Bernoulli KL divergences in Equation (5.1).

The objective compares marginal distributions block by block. Incoherent current prices have no valid joint distribution with those marginals; even coherent marginals may have several joint distribution witnesses. The sum therefore has no general interpretation as KL between two full joint distributions over Ω . Adding a duplicate priced block changes the relative weight of its marginal comparison.

5.3 The direction follows from whose expectations are being taken

For one binary event, let z be the realized zero-or-one payoff. Define the *logarithmic forecast score* at a quoted probability p by

$$z \log p + (1 - z) \log(1 - p).$$

If the event has probability μ , the expected increase in this score from changing the quote from p to μ is

$$\mu \log \frac{\mu}{p} + (1 - \mu) \log \frac{1 - \mu}{1 - p}.$$

The expectation is taken under the candidate probability μ . That is why μ multiplies the first logarithm and $1 - \mu$ multiplies the second. In LMSR, multiplying this expected score increase by b gives the trader’s best expected profit. Hanson’s scoring-rule construction provides this interpretation [P2].

Reversing the arguments changes which probabilities weight the calculation. The two-contract comparison below calculates the resulting change in the minimizer.

5.4 What the objective prefers

Since $C(\theta)$ is constant during a correction, minimizing F_θ is equivalent to maximizing

$$\theta^\top \mu - R(\mu).$$

Define the *binary entropy* by

$$H(\mu_i) = -\mu_i \log \mu_i - (1 - \mu_i) \log(1 - \mu_i).$$

The equivalent maximization becomes

$$\max_{\mu \in \mathcal{M}} \left[\theta^\top \mu + b \sum_i H(\mu_i) \right].$$

A positive θ_i increases the linear reward for larger μ_i ; a negative θ_i favors smaller μ_i . The entropy term alone is maximized at one half. Coherence constraints couple these coordinate choices. If current prices are already coherent, they minimize F_θ with value zero.

The pricing state is a quantity used by the market's pricing rule. This maximization does not require treating that state as a statistical likelihood or assuming that the correction discovers the true probabilities.

5.5 Why squared distance gives a different answer

Return to two identical contracts with current prices $(0.8, 0.4)$ and $b = 1$. Every coherent candidate has the form (r, r) . Define

$$f(r) = F_\theta(r, r) = \text{KL}(\text{Bern}(r) \parallel \text{Bern}(0.8)) + \text{KL}(\text{Bern}(r) \parallel \text{Bern}(0.4)).$$

For $0 < r < 1$, its derivatives are

$$f'(r) = 2 \log \frac{r}{1-r} - \log(8/3), \quad f''(r) = \frac{2}{r(1-r)} > 0.$$

The unique stationary point therefore minimizes the convex objective on $[0, 1]$ and gives

$$r = \frac{\sqrt{8/3}}{1 + \sqrt{8/3}} \approx 0.620204.$$

Minimizing $(r - 0.8)^2 + (r - 0.4)^2$ gives $r = 0.6$. For reversed KL,

$$\frac{d}{dr} \text{KL}(\text{Bern}(p) \parallel \text{Bern}(r)) = \frac{r - p}{r(1-r)}.$$

Summing this derivative at $p = 0.8$ and $p = 0.4$ gives $(2r - 1.2)/[r(1-r)]$, negative below 0.6 and positive above it. Thus reversed KL also has its unique minimum at 0.6. Only the candidate-first expression is the expected-profit value for the unchanged LMSR charges.

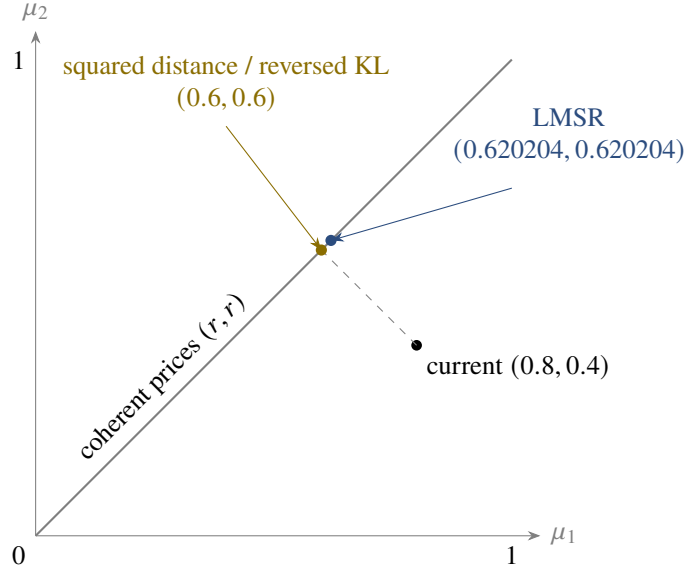


Figure 5.1: Two identical contracts with current prices $(0.8, 0.4)$ and $b = 1$. The feasible segment is the same for all three objectives; candidate-first KL gives the LMSR correction, while squared distance and reversed KL give the common price 0.6.

5.5.1 Optional: local quadratic approximation

Near an interior current price vector, a Taylor expansion makes the difference visible:

$$F_{\theta}(p + h) = \frac{b}{2} \sum_i \frac{h_i^2}{p_i(1 - p_i)} + O(\|h\|^3),$$

where h is small enough that $p + h \in (0, 1)^d$. The objective is defined on the entire interior of $[0, 1]^d$ before the coherent-price constraint is imposed, so this expansion also applies at an incoherent p . The quadratic approximation weights changes by $1/[p_i(1 - p_i)]$. It is not an unweighted squared distance. This approximation is local; it is not a substitute for the full objective on large corrections.

5.6 Liquidity scaling and constants

Common liquidity. Holding the current prices fixed, multiplying every market's liquidity by the same factor multiplies the objective and profit values by that factor. It leaves the minimizing candidate unchanged. Equivalently, at fixed prices p , $F_{b,p}(\mu) = bF_{1,p}(\mu)$. Holding the pricing state fixed instead changes the current prices, so it is a different comparison. Different liquidity parameters across markets create different relative weights.

Constants. Omitting $C(\theta)$ does not change the optimizer. It does change the objective's numerical value. Restore that constant before interpreting the value as profit or using a stopping certificate.

5.7 Optional: a market whose objective is squared distance

For comparison, choose another convex conjugate:

$$\tilde{R}(\mu) = \frac{b}{2} \|\mu\|^2 \quad \text{on } [0, 1]^d,$$

and set it to $+\infty$ elsewhere. Define its cost function by

$$\tilde{C}(\theta) = \max_{\mu \in [0, 1]^d} \left[\theta^\top \mu - \frac{b}{2} \|\mu\|^2 \right].$$

Within this comparison, define the current price vector by $\tilde{p} = \nabla \tilde{C}(\theta)$. Each current price is θ_i/b , restricted to the interval $[0, 1]$. If every current price lies strictly inside that interval, then $\tilde{C}(\theta) = \|\theta\|^2/(2b)$. Restriction to $[0, 1]$ is the exact gradient of this cost, including its flat-price regions; it is not an approximation obtained by clipping LMSR prices.

When every current price satisfies $0 < \tilde{p}_i < 1$, substituting into the correction-objective definition gives, for $\mu \in [0, 1]^d$,

$$\tilde{F}_\theta(\mu) = \frac{b}{2} \|\mu - \tilde{p}\|^2.$$

At these interior current states, the correction objective is squared distance. To compare mechanisms at the same initial price p_i , initialize LMSR at $\theta_i = b \log(p_i/(1 - p_i))$ and the alternative at $\tilde{\theta}_i = bp_i$.

Exercises.

1. With $b = 1$, initialize one LMSR contract at $\theta = 0$ and one quadratic-cost contract at $\tilde{\theta} = 1/2$. Both quote $1/2$. Buy $1/4$ share from each, compute the two cost differences, and explain why equal initial quotes permit different finite charges.
2. Repeat the identical-contract correction with common liquidity $b = 2$ while holding current prices at $(0.8, 0.4)$. Determine which of the corrected prices, trade quantities, and guaranteed profits change.

Chapter 6

Bregman projection and guaranteed profit

Derive the best guaranteed-profit value and construct an attaining trade for interior binary LMSR prices.

6.1 Guaranteed profit is a different optimization

Assume a finite nonempty payoff set $\mathcal{Z} \subseteq \{0, 1\}^d$, coherent set $\mathcal{M} = \text{conv}(\mathcal{Z})$, unrestricted signed trades, and a finite convex cost C on $\mathbb{R}^d$. Its conjugate R is finite on $\mathcal{Z}$. The expected-profit value is $F_\theta(\mu) = R(\mu) + C(\theta) - \theta^\top \mu$.

A trade's *guaranteed profit* is its smallest profit over valid outcomes. The best guaranteed-profit value is

$$A(\theta) = \sup_{\delta \in \mathbb{R}^d} \min_{z \in \mathcal{Z}} [\delta^\top z - C(\theta + \delta) + C(\theta)].$$

The equality of optimal values is

$$A(\theta) = \min_{\mu \in \mathcal{M}} F_\theta(\mu). \quad (6.1)$$

The left side chooses a trade; the right side chooses coherent expected payoffs. Both values are in dollars [P1, Proposition 2.4 and Appendix A]. Equality does not assert the existence of a finite maximizing trade. We first construct one for an interior binary-LMSR minimizer, then establish the general value identity. The stronger claim that every inverse-price state attains the value requires qualification; the optional source note in Section 6.7 gives a counterexample.

6.2 Every coherent candidate provides an upper bound

Fix any coherent μ and any trade δ . The smallest settlement payoff cannot exceed its expectation under a distribution producing μ :

$$\min_{z \in \mathcal{Z}} \delta^\top z \leq \delta^\top \mu.$$

Subtracting the trade charge gives

$$\begin{aligned} \min_{z \in \mathcal{Z}} [\delta^\top z - C(\theta + \delta) + C(\theta)] &\leq \delta^\top \mu - C(\theta + \delta) + C(\theta) \\ &\leq F_\theta(\mu). \end{aligned}$$

The second inequality follows because $F_\theta(\mu)$ optimizes expected profit over all trades. Taking the supremum over trades gives

$$A(\theta) \leq \min_{\mu \in \mathcal{M}} F_\theta(\mu).$$

Each coherent candidate supplies an upper bound on every trade's guaranteed profit. Minimization selects the smallest such bound. The distribution witness supports this inequality; the argument does not estimate the true outcome probabilities.

6.3 The projection problem

Choose a minimizer

$$\mu^\star \in \arg \min_{\mu \in \mathcal{M}} F_\theta(\mu). \quad (6.2)$$

The conjugate is lower semicontinuous and finite on the compact coherent set, so a minimizer exists. The paper calls this the *Bregman projection*, writing its objective as $D(\mu \parallel \theta)$ instead of $F_\theta(\mu)$.

6.4 For interior binary LMSR, the smallest upper bound is attained

Specialize to separate binary LMSR costs,

$$C(\theta) = b \sum_i \log(1 + e^{\theta_i/b}), \quad b > 0.$$

Suppose the minimizer satisfies $0 < \mu_i^\star < 1$ for every coordinate. Then

$$\hat{\theta}_i = b \log \frac{\mu_i^\star}{1 - \mu_i^\star}$$

is finite and quotes $\mu^\star$. Define $\delta^\star = \hat{\theta} - \theta$. Differentiating the correction objective gives

$$\nabla F_\theta(\mu^\star) = \hat{\theta} - \theta = \delta^\star.$$

Because $\mu^\star$ minimizes a differentiable convex function over $\mathcal{M}$,

$$(\delta^\star)^\top (\mu - \mu^\star) \geq 0 \quad (\mu \in \mathcal{M}).$$

Every valid payoff belongs to $\mathcal{M}$, so this first-order condition gives

$$\min_{z \in \mathcal{Z}} (\delta^\star)^\top z \geq (\delta^\star)^\top \mu^\star.$$

The mixture representation $\mu^\star = \sum_z \lambda_z z$ gives the reverse:

$$\min_{z \in \mathcal{Z}} (\delta^\star)^\top z \leq \sum_z \lambda_z (\delta^\star)^\top z = (\delta^\star)^\top \mu^\star.$$

Thus worst and expected settlement payoffs coincide. Subtracting the common charge, and using conjugacy at $\hat{\theta}$, yields

$$\begin{aligned} \min_z (\delta^\star)^\top z - C(\hat{\theta}) + C(\theta) &= (\hat{\theta} - \theta)^\top \mu^\star - C(\hat{\theta}) + C(\theta) \\ &= R(\mu^\star) - \theta^\top \mu^\star + C(\theta) = F_\theta(\mu^\star). \end{aligned}$$

This finite trade attains the upper bound and proves [Equation \(6.1\)](#) for the interior binary-LMSR case.

6.4.1 General value identity and finite attainment

For the general finite convex cost, write $h(\delta, \mu) = \delta^\top \mu - C(\theta + \delta) + C(\theta)$. The trade domain $\mathbb{R}^d$ is convex, and $\mathcal{M}$ is compact and convex. The function h is continuous, concave in the trade, and affine in the candidate. Sion's minimax theorem therefore gives

$$\begin{aligned} A(\theta) &= \sup_{\delta \in \mathbb{R}^d} \min_{\mu \in \mathcal{M}} h(\delta, \mu) \\ &= \min_{\mu \in \mathcal{M}} \sup_{\delta \in \mathbb{R}^d} h(\delta, \mu) = \min_{\mu \in \mathcal{M}} F_\theta(\mu). \end{aligned}$$

A linear function has the same minimum over $\mathcal{Z}$ and its convex hull, which gives the first equality. This is the value argument of Appendix A. It proves [Equation \(6.1\)](#) without asserting finite attainment. Forced zero-or-one prices require the settlement treatment in [Section 13.2](#).

An equivalent expression uses the constrained conjugate cost

$$C_{\mathcal{M}}(\theta) = \sup_{\mu \in \mathcal{M}} \{\theta^\top \mu - R(\mu)\}, \quad A(\theta) = C(\theta) - C_{\mathcal{M}}(\theta).$$

The excess of the base cost over this constrained cost is the available arbitrage value. This identity does not require the mechanism to charge $C_{\mathcal{M}}$ on each real trade; [Chapter 14](#) describes its actual trade and correction steps.

6.5 Work through the identical-contract example

Suppose two contracts pay on the same event. Then

$$\mathcal{Z} = \{(0, 0), (1, 1)\}, \quad \mathcal{M} = \{(r, r) : 0 \leq r \leq 1\}.$$

Use $b = 1$, prices $p = (0.8, 0.4)$, and state $\theta = (\log 4, \log(2/3)) \approx (1.386294, -0.405465)$. The scalar objective $f(r) = F_\theta(r, r)$ and its derivatives were derived in [Section 5.5](#). Its unique minimum satisfies

$$2 \log \frac{r}{1-r} = \log(8/3), \quad r = \frac{\sqrt{8/3}}{1 + \sqrt{8/3}} \approx 0.620204.$$

The projected prices are $\mu^\star = (r, r)$, with

$$\widehat{\theta} = \left(\frac{1}{2} \log(8/3), \frac{1}{2} \log(8/3) \right) \approx (0.490415, 0.490415).$$

The trade sells the first contract and buys the same quantity of the second:

$$\delta^\star = \widehat{\theta} - \theta \approx (-0.895880, 0.895880).$$

Quantity	Value at $b = 1$	Interpretation
Current cost $C(\theta)$	2.120264	Used to calculate charge
New cost $C(\widehat{\theta})$	1.936243	Used to calculate charge
Charge to trader	-0.184021	Immediate receipt of 0.184021
Payoff if the event fails	0	Both contracts pay zero
Net payoff if it occurs	0	Equal long and short payoffs cancel
Guaranteed profit	0.184021	Same in both valid outcomes
Minimum objective	0.184021	Equals guaranteed profit

The table evaluates the analytic solution at $b = 1$; displayed values are rounded to six decimals.

6.6 Coherence does not imply optimal profit extraction

The coherent candidate $(0.6, 0.6)$ has objective value approximately 0.185743, above the minimum 0.184021. Its inverse-price trade receives $\log(4/3) \approx 0.287682$ immediately and owes $\log(32/27) \approx 0.169899$ if the event occurs:

Outcome	Settlement payoff	Trader profit
Event fails	0	0.287682
Event occurs	-0.169899	0.117783

Its guaranteed profit is $\log(9/8) > 0$. The correction is safe but extracts less than the optimum. Its expected profit under the candidate is 0.185743, exceeding the minimum profit by about 0.067960.

For a coherent interior candidate, [Section 9.4](#) defines the Frank–Wolfe gap $g(\mu)$, and [Section 10.2](#) proves

$$\text{minimum profit of the inverse-price trade} = F_\theta(\mu) - g(\mu).$$

A partially optimized candidate can be used for an internal correction when its certified minimum profit is nonnegative (or meets a specified extraction threshold). The inequality is the acceptance condition; the identity alone is not. An accepted coherent state has coherent prices even when extraction is suboptimal.

6.7 Optional source concordance: Proposition 2.4

The value identity from Proposition 2.4 is valid under the assumptions above. Its stronger assertion about *every* inverse-price state is false for a generic nonsmooth conjugate. The following source statement locates that limitation; the finite-attainment result used here is the interior-LMSR proof in [Section 6.4](#).

Proposition 2.4 of the paper (proof in Appendix A). *If the market is in a state θ , the guaranteed profit of any trader is at most $D(\mu^* \parallel \theta)$ where μ^* is the Bregman projection of θ on $\mathcal{M}$. Furthermore, this profit is achieved by any trade δ^* moving the market to a state θ^* with $p(\theta^*) = \mu^*$.*

The paper's D is the tutorial's F_θ ; its μ^* is the minimizer in Equation (6.2). Appendix A infers from a constrained minimum that $u \cdot (z - \mu^*) \geq 0$ for every $u \in \partial F(\mu^*)$. For a nonsmooth objective, a suitable supporting vector is needed; an arbitrary subgradient need not satisfy this inequality. The unique gradient in the interior-LMSR proof avoids that inference.

For a one-dimensional counterexample, let $\mathcal{Z} = \{0, 1\}$ and

$$R(\mu) = \frac{1}{2}\mu^2 + |\mu - \frac{1}{2}|, \quad \mu \in [0, 1],$$

with $R = +\infty$ elsewhere, and let $C = R^*$. Strong convexity of R makes C differentiable. At $\mu = 1/2$,

$$\partial R(1/2) = [-1/2, 3/2].$$

For every θ in this interval, the subgradient relation says that $\mu = 1/2$ maximizes the conjugate defining C . Since $R(1/2) = 1/8$,

$$C(\theta) = \theta/2 - 1/8 \quad (-1/2 \leq \theta \leq 3/2).$$

Thus $\theta = 1/2$ and $\hat{\theta} = 3/2$ both quote the coherent price $1/2$. The initial projection value is zero, yet

$$C(1/2) = 1/8, \quad C(3/2) = 5/8.$$

The trade $\Delta = 1$ costs $1/2$ and has worst payoff zero, so its guaranteed profit is $-1/2$ rather than the optimal value zero.

For the paper's entropy-gradient construction, surviving positive categorical LMSR coordinates have inverse-price states differing only by blockwise cash-equivalent constants. Those constants cancel in profit, so this counterexample leaves that construction intact. An extension to a new cost function must verify the supporting-vector and inverse-state properties it uses [P1, Appendix A compared with Appendix C].

Exercises.

1. For the target $(0.6, 0.6)$, derive the inverse-price trade and both outcome profits from the cost differences. Check its expected profit using weights 0.4 and 0.6 on the two outcomes.
2. For the nonsmooth counterexample, calculate $\partial F_{1/2}(1/2)$. Find one member satisfying the supporting inequality for both $z = 0, 1$ and one member that fails it.

Chapter 7

The loss budget

Bound settlement payments minus collected cash, then compute the budget for uniform and nonuniform LMSR initialization.

7.1 Define the loss before bounding it

How much can settlement payments exceed the cash collected from traders? Let q denote aggregate shares held by outside traders and K the net cash collected from them. A valid payoff vector $z \in \mathcal{Z}$ lists the payment per share at settlement. The *market-maker loss* is

$$L(z) = q^\top z - K.$$

A positive value means payments exceed receipts. The quantity $L(z)$ uses actual holdings and cash. The correction objective $F_\theta(\mu)$ is a hypothetical trader's best expected-profit value; it enters the budget formula but is not an account balance.

Assume a fixed, finite nonempty payoff set $\mathcal{Z} \subseteq \{0, 1\}^d$ and a fixed finite convex differentiable cost $C : \mathbb{R}^d \rightarrow \mathbb{R}$. Real signed trades are unrestricted and pay the cost difference in Equation (2.1), without additional fees [P1, §2.1]. The initial proof assumes no internal corrections. Their effect on the accounts is treated in Section 14.2; changes at settlement are treated in Section 13.2.

7.2 Derive the loss budget

Initially assume $q = 0$, $K = 0$, pricing state θ^0 , and no internal corrections. The initial state sets starting quotes; it need not represent shares already owned by traders. For two successive real trades, cash receipts telescope as

$$[C(\theta^1) - C(\theta^0)] + [C(\theta^2) - C(\theta^1)] = C(\theta^2) - C(\theta^0).$$

With a fixed cost and no internal state changes, the same cancellation applies to any trading sequence:

$$q = \theta - \theta^0, \quad K = C(\theta) - C(\theta^0).$$

Therefore, for a fixed payoff vector,

$$L(z) = (\theta - \theta^0)^\top z - C(\theta) + C(\theta^0).$$

The worst loss at that payoff vector is the supremum over the final pricing state. Collect the terms that depend on that state:

$$\begin{aligned}\sup_{\theta} L(z) &= \sup_{\theta} \{\theta^\top z - C(\theta)\} - (\theta^0)^\top z + C(\theta^0) \\ &= R(z) - (\theta^0)^\top z + C(\theta^0).\end{aligned}$$

The last equality uses the convex conjugate $R(z) = \sup_{\theta} \{\theta^\top z - C(\theta)\}$. Unrestricted trades allow every finite final state.

Assume $R(z) < \infty$ for each $z \in \mathcal{Z}$. Taking the maximum over the finite valid payoff set gives the *loss budget*

$$B(\theta^0) = \max_{z \in \mathcal{Z}} \{R(z) + C(\theta^0) - (\theta^0)^\top z\}. \quad (7.1)$$

Consequently $L(z) \leq B(\theta^0)$ for every allowed trading sequence and settlement. This is an exact worst-case supremum under the stated model; a finite trade need not attain it [P1, Appendix B].

This is Proposition B.1 of the paper, with $z = \phi(\omega)$ and $D(z|\theta^0) = F_{\theta^0}(z)$ [P1, Appendix B]. Finiteness of the conjugate on the valid payoffs makes the budget finite.

7.3 Compute the budget for LMSR

For d binary logarithmic market scoring rule (LMSR) markets with liquidity $b > 0$,

$$C(\theta) = b \sum_{i=1}^d \log(1 + e^{\theta_i/b}),$$

$$R(\mu) = b \sum_{i=1}^d \{\mu_i \log \mu_i + (1 - \mu_i) \log(1 - \mu_i)\} \quad \text{on } [0, 1]^d.$$

Use $0 \log 0 = 0$ and set $R = +\infty$ outside this cube. Every valid payoff coordinate equals zero or one, so $R(z) = 0$. Uniform initialization means $\theta^0 = 0$, giving

$$B(0) = db \log 2.$$

For example, two binary markets with $b = 1$ have budget $2 \log 2 \approx 1.386294$ in the model's currency units. At uniform initialization, logical dependence does not change this bound because every valid payoff has $R(z) = 0$ and $\theta^0 = 0$. For nonuniform initialization, the term $-(\theta^0)^\top z$ varies with the valid joint payoff.

The budget bounds potential losses. It does not say that traders have already paid $C(0)$ or that the market has that amount in its cash account. The initial cash account here is zero. Choosing larger b increases this budget proportionally at fixed initial prices.

7.3.1 A budget and a current cash balance

Start one binary market at $b = 1$ and $\theta^0 = 0$, with $q = K = 0$. Its budget is $\log 2 \approx 0.693147$. After a trader buys one share, the market holds cash

$$K = C(1) - C(0) = \log(1 + e) - \log 2 \approx 0.620115$$

and owes one share. If the event occurs, its loss is $1 - K \approx 0.379885$; if it fails, its loss is $-K \approx -0.620115$. The current worst loss is 0.379885, below the budget, while the collected cash is 0.620115.

For a purchase of $q > 0$ shares at general liquidity b , the loss in the paying outcome is

$$q - b \log(1 + e^{q/b}) + b \log 2 = b \log 2 - b \log(1 + e^{-q/b}).$$

It approaches $b \log 2$ as $q \rightarrow \infty$ and is strictly smaller for every finite q . The budget is an exact supremum even when no finite trade attains it.

7.3.2 Categorical blocks and nonuniform initialization

For the paper's compositional cost, a sum of categorical LMSR blocks each of the form (2.3), every payoff block is one-hot, so $R(\phi(\omega)) = 0$. At uniform initialization $\theta^0 = 0$,

$$B(0) = \sum_j b \log |X_j|.$$

For the reduced binary model, this becomes $db \log 2$, as above.

For each complete categorical block, let initial prices $p_{j,x}^0 > 0$ sum to one and choose $\theta_{j,x}^0 = b \log p_{j,x}^0$. Its cost is $b \log \sum_x p_{j,x}^0 = 0$, so $C(\theta^0) = 0$ and

$$B(\theta^0) = \max_{\omega \in \Omega} \sum_j -b \log p_{j,X_j(\omega)}^0,$$

where $X_j(\omega)$ is the category of block j in outcome ω . For a reduced binary coordinate, the equivalent relative state is instead $\theta_i^0 = b \log(p_i^0/(1 - p_i^0))$. Its unshifted scalar cost need not vanish; the removed No-state component accounts for the difference.

Derivation. The maximum over compatible outcomes in the nonuniform categorical budget above can be computed with the outcome IP (Chapter 8). Summing each block's worst outcome gives a valid upper bound; incompatible worst outcomes need not occur jointly. For a strict inequality, offer two contracts on the *same* event, initialized at Yes probabilities 0.8 and 0.4, with $b = 1$. The first block's least likely category is No, with probability 0.2; the second's is Yes, with probability 0.4. These categories cannot occur together. The exact budget is

$$\max\{-\log(0.8 \cdot 0.4), -\log(0.2 \cdot 0.6)\} = -\log 0.12 \approx 2.120264,$$

whereas summing separate maxima gives $-\log 0.08 \approx 2.525729$.

Adding a market changes the loss-budget calculation. Record the new block's initialization and the additional subsidy bound before offering it.

7.3.3 Changing the cost normalization changes no charges

For any constant a , define $\tilde{C} = C + a$. Then $\tilde{R} = R - a$. Charges, prices, the correction objective, and the loss budget remain unchanged because the added constants cancel. Thus setting $C(\theta^0) = 0$ by normalization does not eliminate the subsidy requirement.

Exercises.

1. Start one binary LMSR market at price 0.8 with $b = 1$ and no outside holdings or collected cash. Find its initial state and loss budget. After a one-share purchase, calculate collected cash and the market's loss in each outcome.
2. For the two identical contracts above, derive the same nonuniform budget from reduced binary logits and from complete categorical states. Account for the cost-normalization constants in each representation.

Part III

The algorithm

Chapter 8

Outcome spaces as integer programs

Build and audit the integer payoff model, using the triangle example to distinguish local consistency from a feasible joint distribution.

8.1 What must be exact

Paper. For the contract systems modeled here, logical structure can give a compact integer formulation. Compactness is a modeling property to establish; it is not automatic for every subset of the binary cube. In its simplest form, the paper assumes  §2.3, Eq. (2)

$$\mathcal{Z} = \{z \in \{0, 1\}^d : Az \geq a\}. \quad (8.1)$$

Here A uses rows for constraints; the paper writes the equivalent convention $\mathbf{A}^\top z \geq \mathbf{b}$ [P1, Eq. (2)]. The letter b is reserved for liquidity in this document.

Implementation recommendation. Auxiliary variables are often convenient:

$$\mathcal{Z} = \{z \in \{0, 1\}^d : \exists w \text{ such that } (z, w) \in \mathcal{K}\},$$

where $\mathcal{K}$ is a mixed-integer feasible set. The oracle objective (Equation (9.1)) gives zero coefficients to purely auxiliary coordinates. The convex combination and entropy objective use payoff coordinates z , not arbitrary modeling variables w . Deduplicate columns by their payoff vector, not by auxiliary assignments.

Check both directions of correctness:

1. Every real outcome yields a feasible IP point with the correct payoffs.
2. Every feasible IP payoff vector corresponds to a real possible outcome.

The first prevents excluding genuine outcomes. The second prevents allowing fictitious outcomes. Testing only that a few simulated outcomes satisfy the constraints establishes neither equality of sets nor the required guarantee. Excluding a real outcome can make a minimum-profit guarantee unsafe. Admitting fictitious outcomes can make profit bounds conservative, but mixtures using them need not be genuinely coherent.

8.2 Standard logical constraints

For binary variables, the following formulations are exact:

Relation	Constraints
$y = x_1 \wedge x_2$	$y \leq x_1, \quad y \leq x_2, \quad y \geq x_1 + x_2 - 1$
$y = x_1 \vee x_2$	$y \geq x_1, \quad y \geq x_2, \quad y \leq x_1 + x_2$
$y = \neg x$	$y = 1 - x$
Exactly one outcome in a category	$\sum_{x \in \mathcal{X}_j} z_{j,x} = 1$
At most one event	$\sum_{i \in S} z_i \leq 1$
Implication $x \Rightarrow y$	$x \leq y$

Bounds and integrality are part of these statements. Replacing integrality with $[0, 1]$ bounds gives an outer relaxation, which can admit candidate price vectors with no joint interpretation (Section 8.6).

For the paper's two-team example, each win count lies in $\{0, \dots, 6\}$, and both teams having at least five wins is forbidden because they meet in round five. Binary one-hot indicators satisfy

$$\sum_{r=0}^6 z_{j,r} = 1 \quad (j = 1, 2), \quad z_{1,5} + z_{1,6} + z_{2,5} + z_{2,6} \leq 1.$$

Every allowed count pair satisfies these rows. For the converse, the binary one-hot rows select exactly one count per team; the final inequality excludes exactly the pairs where both counts are at least five. Every remaining pair belongs to the stated outcome space [P1, Examples 2.2 and 2.5].

8.3 The six-security triangle example

Let the underlying outcome be $(x_1, x_2, x_3) \in \{0, 1\}^3$. Trade six Yes securities with payoff vector

$$z = (x_1, x_2, x_3, y_{12}, y_{23}, y_{13}), \quad y_{ij} = x_i x_j.$$

There are eight feasible payoff vectors. Enforce each product with the three AND inequalities of Section 8.2. The rows below follow the starter code's lexicographic bit order and its security order. The last column is $\ell(z) = x_1 + x_2 + x_3 - y_{12} - y_{23} - y_{13}$.

x_1	x_2	x_3	y_{12}	y_{23}	y_{13}	$\ell(z)$
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	0	1
0	1	1	0	1	0	1
1	0	0	0	0	0	1
1	0	1	0	0	1	1
1	1	0	1	0	0	1
1	1	1	1	1	1	0

Now consider the price vector

$$p^0 = (0.5, 0.5, 0.5, 0.05, 0.05, 0.05).$$

It satisfies every individual pair's McCormick/AND relaxation. For each pair, the four cells follow from

$$P_{11} = p_{ij}, \quad P_{10} = p_i - p_{ij}, \quad P_{01} = p_j - p_{ij}, \quad P_{00} = 1 - p_i - p_j + p_{ij}.$$

The implied four-entry table is $(P_{00}, P_{10}, P_{01}, P_{11}) = (0.05, 0.45, 0.45, 0.05)$, which is a legitimate pairwise distribution and has the specified one-variable marginals.

But every binary outcome satisfies

$$x_1 + x_2 + x_3 - y_{12} - y_{23} - y_{13} \leq 1.$$

To see this, if k underlying variables equal one, the left side is $k - \binom{k}{2}$, which equals 0, 1, 1, 0 for $k = 0, 1, 2, 3$. The proposed prices instead give $1.5 - 0.15 = 1.35$.

Thus each pairwise table is valid, but the three tables cannot arise from one distribution on all three binary variables. This is exactly the distinction the full IP oracle resolves.

The initial arbitrage direction is also explicit:

$$\delta = t(-1, -1, -1, 1, 1, 1), \quad t > 0.$$

Its worst payoff is $-t$, while its first-order cost at p^0 is $-1.35t$. For sufficiently small t , the trade therefore guarantees positive profit, approximately $0.35t$ before accounting for curvature. The example returns in [Chapters 15](#) and [16](#).

8.4 A tournament formulation you can build incrementally

Implementation recommendation. Use a fixed bracket with $N = 2^k$ teams. For each actual game g , let T_g be the teams in its subtree and introduce

$$w_{g,t} = \mathbf{1}\{t \text{ wins game } g\}, \quad t \in T_g.$$

Require $w_{g,t} \in \{0, 1\}$. Impose $\sum_{t \in T_g} w_{g,t} = 1$ for each game. If team t must win child game h to reach parent game g , impose

$$w_{g,t} \leq w_{h,t}.$$

This recursively forces every game winner to be one of the winners who reach it. There are $N - 1$ games and Nk winner indicators when all potential teams at each round are represented.

For each team, add binary one-hot win-count indicators $v_{t,r} = \mathbf{1}\{t \text{ wins exactly } r \text{ games}\}$, $r = 0, \dots, k$, with

$$\sum_{r=0}^k v_{t,r} = 1, \quad \sum_{r=0}^k r v_{t,r} = \sum_{g: t \in T_g} w_{g,t}.$$

Require $v_{t,r} \in \{0, 1\}$ as well. For four teams, let game a pair teams 1 and 2, game b pair teams 3 and 4, and game f be the final. The one-winner equations and

$$w_{f,t} \leq w_{a,t} \quad (t = 1, 2), \quad w_{f,t} \leq w_{b,t} \quad (t = 3, 4)$$

select two semifinal winners and one of them as champion. Every bracket outcome yields such an assignment, and every assignment describes a bracket outcome. Represent each actual game once, including the final.

8.4.1 Two encodings, different relaxations

At integer points this is an exact alternative to the paper’s difference-of-round-survival equations. It can have a different LP relaxation, so do not treat it as the same LCMM baseline without checking [P1, §3.1].

For the paper’s formulation, define $s_{t,r}$ as the indicator that team t wins round r , let $s_{t,0} = 1$, and set $s_{t,k+1} = 0$. Then

$$v_{t,r} = s_{t,r} - s_{t,r+1}, \quad r = 0, \dots, k.$$

The source formulation is recorded in [Section 8.7](#). Represent each actual game once. Do not accidentally create independent copies of the same game for each team that could reach it.

8.5 Sums and comparisons

For a sum variable $S = \sum_j X_j$ with categorical payoff indicators, impose

$$\sum_s s z_{S,s} = \sum_j \sum_x x z_{j,x}, \quad \sum_s z_{S,s} = 1.$$

For a comparison $Q \in \{\text{lt}, \text{eq}, \text{gt}\}$ of integer-valued U and V , use one-hot comparison indicators and the implications

$$z_{Q,\text{lt}} = 1 \Rightarrow U \leq V - 1,$$

$$z_{Q,\text{eq}} = 1 \Rightarrow U = V,$$

$$z_{Q,\text{gt}} = 1 \Rightarrow U \geq V + 1.$$

Here U and V are the linear expressions associated with their one-hot variables. Use native indicator constraints if the selected solver interface supports them, or derive big- M values from the actual bounds of $U - V$. The AND-based starter implementation in [Chapter 16](#) avoids big- M entirely.

8.5.1 Heuristic initial prices

The paper initializes sums and differences with discretized Gaussian approximations using marginal means and sums of marginal variances [P1, §3.1]. This is a heuristic, separate from the exact payoff model: the variance of a sum generally includes covariance terms. Marginal prices do not determine a unique joint distribution.

For $Y = X_2 - X_1$, the correct mapping is lt for $Y > 0$, eq for $Y = 0$, and gt for $Y < 0$. For instance, $X_1 = 1, X_2 = 2$ gives $Y = 1$ and category lt. The manuscript’s comparison-price initialization contains an apparent sign reversal: it defines $Y = X_2 - X_1$ but associates lt/eq/gt with $Y < 0$, $Y = 0$, and $Y > 0$. With that definition, lt must correspond to $Y > 0$ and gt to $Y < 0$. This was checked against the rendered PDF, p. 8 of the paper. It concerns the initialization paragraph, not the definition of the comparison itself. Test comparison labels with explicit ordered pairs before replaying trades [P1, §3.1].

8.6 LCMM as an outer approximation

The cheaper layer maintains an outer polytope

$$\mathcal{M} \subseteq \widetilde{\mathcal{M}} = \{\mu : H\mu \geq h\}. \quad (8.2)$$

An LCMM correction uses $\Delta = H^\top \eta$ for $\eta \geq 0$. Since $H\bar{z} \geq h$, multiplying by $\eta^\top$ gives $\Delta^\top \bar{z} = \eta^\top H\bar{z} \geq \eta^\top h$. Every real outcome satisfies

$$\Delta^\top \bar{z} \geq h^\top \eta.$$

Therefore, if the correction also satisfies

$$C(\theta + H^\top \eta) - C(\theta) \leq h^\top \eta,$$

it is a nonnegative-profit internal trade. Optimizing $C(\theta + H^\top \eta) - h^\top \eta$ over $\eta \geq 0$ is one way to find such corrections. A partially solved correction can be safe without enforcing every inequality; check what each return status actually certifies. The LCMM step of Mechanism 1 of the paper (Mechanism 1) chooses η by the same condition.

These acceptance inequalities are the construction in the paper's Section 2.4 [P1, §2.4]. For the triangle example, the pairwise LP relaxation accepts p^0 unchanged. Adding the single triangle inequality exposes the problem. This is an excellent first demonstration of the relationship between LCMM cuts and the IP oracle.

8.7 Optional source concordance

The model-building argument is complete. Continue to the exercises or to Chapter 9 for the optimization algorithm; the restatements below are reference material for comparing the formulations with the paper. The source's Eq. (2) is (8.1) after transposing its constraint matrix, and Eq. (3) is the box-bounded LP relaxation. The structured models here admit compact formulations; compactness is not a universal consequence of integrality. The paper's $\mathcal{I}$ indexes the d security coordinates.

The sum/comparison excerpt below retains the source's heuristic initialization and its apparent sign reversal. With $Y = X_2 - X_1$, It requires $Y > 0$ and gt requires $Y < 0$. This warning applies before using the excerpt. The comparison relations are inequalities, even where the source calls them identities.

Paper: Compositional design and atomic tournament variables, §3.1. The market construction begins with a collection of random variables $X_j : \Omega \rightarrow \mathcal{X}_j$, indexed by $j \in \mathcal{J}$, whose marginal distributions are to be elicited. Securities are indexed by $i = (j, x)$, with $j \in \mathcal{J}$ and $x \in \mathcal{X}_j$, and correspond to indicators of the events $X_j = x$, i.e., $\phi_{j,x}(\omega) = \mathbf{1}\{X_j(\omega) = x\}$. The cost function is the sum of LMSRs across the random variables X_j (the paper's Eq. (4)). The notation $z\{X_j = x\}$ and $\mu\{X_j = x\}$ stands for the entries $z_{j,x}$ and $\mu_{j,x}$. New IP constraints always include the exclusivity and exhaustivity constraint $\sum_{x \in \mathcal{X}} z\{X = x\} = 1$, but additional constraints may be needed to correctly describe the mapping $X(\omega)$. LCMM constraints are added as an LP relaxation of IP constraints, with the exception of one variable type (comparison variables).

Atomic tournament variables. These random variables model outcomes in a single-elimination tournament with k rounds and 2^k teams. Teams are numbered 1 through 2^k . In the first round, there are 2^{k-1} games, between teams $2i - 1$ and $2i$, and the resulting 2^{k-1} winners advance to the second round, where again teams are matched in the order of increasing indices, and the winners advance to the next round, and so on. Team t is associated with the random variable X_t whose outcome is the total number of wins of team t , i.e., $X_t = \{0, \dots, k\}$.

There are also random variables corresponding to the games played, with the outcome of each variable being the winner of the corresponding game. For a team t and round r , $G_{r,t}$ denotes the game that team t will play in the r -th round if it advances to that point. This slightly abuses notation, because $G_{r,t}$ and $G_{r,t'}$ can refer to the same game (and hence the same random variable) for distinct t and t' (see Figure 1 of the paper). For instance $G_{k,t} \equiv G_{k,t'}$ for all t, t' , as there is only one game (the final) in round k . The IP constraints relating the entries of z representing game and team variables are

$$\begin{aligned} z\{X_t = r\} &= z\{G_{r,t} = t\} - z\{G_{r+1,t} = t\} \quad \text{for all } t \text{ and } r < k, \\ z\{X_t = k\} &= z\{G_{k,t} = t\} \quad \text{for all } t. \end{aligned}$$

LCMM constraints are just LP relaxations of the above, i.e., they are the same as the IP constraints, with $z\{\cdot\}$ replaced with $\mu\{\cdot\}$. The market operator needs to specify initial prices $\mu\{X_t = r\}$ and $\mu\{G_{k,t} = t\}$ explicitly, based for instance on the past performance of teams.

Paper: Sums and comparisons, §3.1. Sums. Given a set of existing random variables $X_1, \dots, X_n$ taking on integer values with the minimum and maximum values $m_j := \min X_j$ and $M_j := \max X_j$, a new random variable X represents their sum,

$$X(\omega) := X_1(\omega) + \dots + X_n(\omega),$$

with the domain $\mathcal{X} = \{m, m+1, \dots, M\}$ where $m = \sum_{j=1}^n m_j$ and $M = \sum_{j=1}^n M_j$. The initial prices are set proportional to a discretized Gaussian distribution with the mean and variance equal to the sum of means and variances of X_1 through X_n , under the distribution described by the current prices $\mu\{X_j = x\}$. The IP constraint is

$$\sum_{x \in \mathcal{X}} x \cdot z\{X = x\} = \sum_{j=1}^n \sum_{x_j \in \mathcal{X}_j} x_j \cdot z\{X_j = x_j\}.$$

The added LCMM constraint is an LP relaxation of the added IP constraint.

Comparisons. Given two existing random variables X_1 and X_2 taking on integer values with the minimum and maximum values $m_j := \min X_j$ and $M_j := \max X_j$, a new random variable X with the domain $\{lt, eq, gt\}$ represents the result of their comparison:

$$X(\omega) := \begin{cases} lt & \text{if } X_1(\omega) < X_2(\omega), \\ eq & \text{if } X_1(\omega) = X_2(\omega), \\ gt & \text{if } X_1(\omega) > X_2(\omega). \end{cases}$$

The initialization prices are determined by first considering an integer-valued variable $Y = X_2 - X_1$, and initializing its distribution to the discrete Gaussian with the mean equal to the difference of means and the variance initialized to the sum of variances of X_2 and X_1 under current prices. The initial prices of $X = lt$, $X = eq$ and $X = gt$ are obtained as probabilities that $Y < 0$, $Y = 0$ and $Y > 0$. The variable Y is discarded and is not part of the market.

The IP constraints for the new entries of z are based on the following four identities:

$$\begin{aligned} X_1 - X_2 &\geq (m_1 - M_2)\mathbf{1}\{X_1 < X_2\}, & X_1 - X_2 - 1 &\geq (m_1 - M_2 - 1)\mathbf{1}\{X_1 \leq X_2\}, \\ X_1 - X_2 &\leq (M_1 - m_2)\mathbf{1}\{X_1 > X_2\}, & X_1 - X_2 + 1 &\leq (M_1 - m_2 + 1)\mathbf{1}\{X_1 \geq X_2\}. \end{aligned}$$

To obtain IP constraints, each X_j on the left-hand side is replaced with $\sum_{x \in X_j} x \cdot z\{X_j = x\}$, and the comparison indicators on the right-hand side are replaced by $z\{X = lt\}$ for $\mathbf{1}\{X_1 < X_2\}$, and $z\{X = lt\} + z\{X = eq\}$ for $\mathbf{1}\{X_1 \leq X_2\}$, and similarly for $X_1 > X_2$ and $X_1 \geq X_2$.

Paper: Linearly-constrained market maker, §2.4 and Eq. (3). The Frank–Wolfe algorithm relies on the ability to solve integer programs (IPs), which can take exponential time in the worst case. Therefore the mechanism also incorporates fast (polynomial-time) partial arbitrage removal similar to the linearly-constrained market maker (LCMM) of Dudík et al. [P4]. In LCMM, arbitrage is partly removed by considering a set of linear constraints that must be satisfied by coherent prices. Namely, an LCMM takes as an input a relaxation $\widetilde{\mathcal{M}} \supseteq \mathcal{M}$ described by linear constraints called *LCMM constraints*:

$$\widetilde{\mathcal{M}} = \{\mu \in \mathbb{R}^I : \widetilde{\mathbf{A}}^\top \mu \geq \widetilde{\mathbf{b}}\}.$$

When any LCMM constraint is violated, there is an arbitrage opportunity in the market, with an easy-to-compute arbitraging trade. LCMM acts as an arbitrager until none of the constraints are violated. Since $\widetilde{\mathcal{M}}$ is a relaxation of $\mathcal{M}$, the resulting state is not necessarily arbitrage-free.

Given a description of $\mathcal{Z}$ using IP constraints specified by a matrix $\mathbf{A}$ and a vector $\mathbf{b}$, one simple strategy is to construct $\widetilde{\mathcal{M}}$ as a linear-program (LP) relaxation of $\mathcal{Z}$, i.e. (the paper’s Eq. (3)),

$$\widetilde{\mathcal{M}} = \{\mu \in \mathbb{R}^I : \mu_i \in [0, 1] \text{ for all } i \in I \text{ and } \mathbf{A}^\top \mu \geq \mathbf{b}\}.$$

These constraints are satisfied by all $z \in \mathcal{Z}$ and hence also by their convex combinations $\mu \in \mathcal{M}$. Generally, this relaxation is only a loose superset of $\mathcal{M}$, so various ad hoc strategies are required to obtain a tighter $\widetilde{\mathcal{M}}$ [P4; P8].

Paper: LCMM constraints for comparisons, §3.1. LCMM constraints for comparison variables are not simply an LP relaxation of IP constraints, but instead yield a tighter set $\widetilde{\mathcal{M}}$. They are based on the following identities, which can be derived from the

transitivity of the comparison and the union bound:

$$\mathbb{P}\{X_1 \leq x\} \leq \mathbb{P}\{X_1 < X_2\} + \mathbb{P}\{X_2 \leq x\} \quad \text{for all } x \geq m_1 \text{ and } x \leq M_2,$$

$$\mathbb{P}\{X_1 \leq x\} \leq \mathbb{P}\{X_1 \leq X_2\} + \mathbb{P}\{X_2 < x\} \quad \text{for all } x \geq m_1 \text{ and } x \leq M_2.$$

For instance, the first inequality follows because $X_1 \leq x$ implies that either $X_1 < X_2$ or $X_2 \leq x$; otherwise there would be a contradiction, $X_1 \geq X_2 > x$. The resulting LCMM constraints are

$$\mu\{X_1 \leq x\} \leq \mu\{X = lt\} + \mu\{X_2 \leq x\} \quad \text{for all } m_1 \leq x \leq M_2,$$

$$\mu\{X_1 \leq x\} \leq \mu\{X \in \{lt, eq\}\} + \mu\{X_2 < x\} \quad \text{for all } m_1 \leq x \leq M_2,$$

with analogous constraints with X_1 and X_2 swapped (and gt swapped for lt). Here $\mu\{X \in \mathcal{E}\}$ is shorthand for $\sum_{x \in \mathcal{E}} \mu\{X = x\}$.

Exercises.

1. Enumerate all 64 binary six-coordinate vectors and check that the three AND formulations admit exactly the triangle table's eight payoffs. Compare oracle minimization and enumeration for $c = (-1, -1, -1, 1, 1, 1)$.
2. Enumerate the eight four-team bracket outcomes and verify both inclusions of the winner/count formulation. Compare its LP relaxation with the survival-difference formulation for one chosen linear objective.
3. Check the comparison mapping for $(X_1, X_2) = (1, 2)$ and $(2, 1)$.

Chapter 9

Frank–Wolfe as column generation

Explain how to minimize the correction objective without listing every possible payoff vector. Implement the nonlinear restricted master and integer pricing oracle, and distinguish a restricted-master solve from a global gap certificate.

We minimize F_θ over coherent prices using a growing set of valid payoff vectors. The obstacle is the size of the outcome set: a useful algorithm must find improving outcomes without listing them all.

9.1 State the problem and its assumptions

Keep the pricing state θ , cost C , and nonempty finite payoff set $\mathcal{Z} \subseteq \{0, 1\}^d$ fixed throughout one correction. The problem is

$$\min_{\mu \in \mathcal{M}} F_\theta(\mu), \quad \mathcal{M} = \text{conv}(\mathcal{Z}), \quad F_\theta(\mu) = R(\mu) + C(\theta) - \theta^\top \mu.$$

Here $R = C^*$, and $b > 0$ denotes liquidity. We use binary LMSR and its conjugate from [Equations \(2.4\) and \(4.4\)](#). Derivatives below require $0 < \mu_i < 1$ on every active binary coordinate. The contraction in [Section 12.4](#) maintains this condition; settled coordinates require the changed cost in [Section 13.2](#).

9.2 The nonlinear master problem

Freeze the pre-correction state θ . The full problem is $\min_{\mu \in \mathcal{M}} F_\theta(\mu)$. If all vertices were enumerated as rows of a matrix $V \in \mathbb{R}^{k \times d}$, with $\lambda \in \mathbb{R}^k$ and $V^\top \lambda \in \mathbb{R}^d$, it could be written

$$\min_{\lambda \geq 0, \mathbf{1}^\top \lambda = 1} F_\theta(V^\top \lambda).$$

At large scale the matrix has too many rows. The term “column generation” names the decomposition method even though the implementation stores payoff vectors as rows. Numerical error in a restricted-master solve and restriction to the active outcomes are separate limitations.

Suppose instead that k possible payoff vectors $z_1, \dots, z_k$ have been found. Choose nonnegative weights $\lambda_1, \dots, \lambda_k$ summing to one and set

$$\mu = \sum_{j=1}^k \lambda_j z_j.$$

This formula proves that $\mu \in \mathcal{M}$. Maintain only the set $\mathcal{Z}_t$ of previously discovered valid payoff vectors and improve the weights by solving the restricted problem

$$\min_{\lambda \in \Delta_{|\mathcal{Z}_t|}} F_\theta\left(\sum_{z \in \mathcal{Z}_t} \lambda_z z\right),$$

where $\Delta_k = \{\lambda \geq 0 : \mathbf{1}^\top \lambda = 1\}$ is the simplex of weights on k active outcomes. This is a convex problem in $|\mathcal{Z}_t|$ variables. Its limitation is that a missing payoff vector might permit further improvement. *Fully corrective Frank–Wolfe* repeatedly finds a useful additional payoff vector and reoptimizes all the active weights after adding a column [P1, §4.1; P5].

Paper. The paper uses accelerated projected gradient for this restricted convex problem [P1, §4.1]. The starter experiment (Chapter 16) uses SciPy’s SLSQP to keep the code short.

The weights establish coherence. They need not uniquely identify a distribution on complete settlements: different weights can produce the same traded prices (Section 9.9).

9.3 The integer pricing oracle

At an interior candidate μ , differentiation gives

$$\frac{\partial F_\theta}{\partial \mu_i}(\mu) = b \log \frac{\mu_i}{1 - \mu_i} - \theta_i.$$

Define the vector

$$c = \nabla F_\theta(\mu) = \nabla R(\mu) - \theta.$$

For a possible payoff vector z , consider moving a small fraction η toward it. The derivative at $\eta = 0$ is

$$\left. \frac{d}{d\eta} F_\theta((1 - \eta)\mu + \eta z) \right|_{\eta=0} = c^\top (z - \mu).$$

The oracle selects the best first-order direction at the current candidate. It does not maximize the actual finite decrease in the nonlinear objective; the subsequent master solve determines the new mixture. Since $c^\top \mu$ is fixed, select the next outcome column

$$z^{\text{new}} \in \arg \min_{z \in \mathcal{Z}} c^\top z, \quad \text{attaining} \quad \min_{z \in \mathcal{Z}} c^\top z. \quad (9.1)$$

This minimization is the *linear oracle*, or integer pricing oracle. It can be implemented as an integer program. For example, if $z = (x_1, x_2, y)$ and y pays when both binary events occur, use

$$y \leq x_1, \quad y \leq x_2, \quad y \geq x_1 + x_2 - 1, \quad x_1, x_2, y \in \{0, 1\}.$$

If either event fails, an upper bound forces $y = 0$; if both occur, the lower bound forces $y = 1$. Thus these constraints describe the four possible payoff vectors exactly (the AND row of Section 8.2). The objective coefficients change between iterations; the payoff rules do not.

Why is a linear oracle sufficient? For any convex combination,

$$c^\top \left(\sum_j \lambda_j z_j \right) = \sum_j \lambda_j c^\top z_j \geq \min_{z \in \mathcal{Z}} c^\top z.$$

Therefore minimizing this linear function over $\mathcal{M}$ is equivalent to minimizing it over $\mathcal{Z}$:

$$\min_{\mu \in \text{conv}(\mathcal{Z})} c^\top \mu = \min_{z \in \mathcal{Z}} c^\top z.$$

The integer outcome model replaces enumeration in this linear minimization. For two identical contracts, take $\mathcal{Z} = \{(0, 0), (1, 1)\}$, current prices $(0.8, 0.4)$, $b = 1$, and candidate $\mu = (0.5, 0.5)$. Then

$$\theta = (\log 4, \log(2/3)), \quad c = (-\log 4, -\log(2/3)) \approx (-1.386294, 0.405465).$$

The two oracle values are 0 and -0.980829 , so the oracle chooses $(1, 1)$. The quantity $c^\top(\mu - (1, 1))$ is approximately 0.490415; the next section explains its role as an error certificate. A direction of improvement does not yet certify a safe state change; that test is derived in [Section 10.2](#).

9.4 The Frank–Wolfe gap

With an exact linear oracle, define the *Frank–Wolfe gap*

$$g(\mu) = c^\top \mu - \min_{z \in \mathcal{Z}} c^\top z = c^\top(\mu - z^{\text{new}}) = \max_{z \in \mathcal{Z}} c^\top(\mu - z). \quad (9.2)$$

It is nonnegative because μ is a convex combination of possible payoff vectors. Let $\mu^\star$ minimize F_θ over $\mathcal{M}$. Convexity gives

$$F_\theta(\mu^\star) \geq F_\theta(\mu) + c^\top(\mu^\star - \mu).$$

Also, $c^\top \mu^\star \geq \min_z c^\top z$, because the oracle minimizes $c^\top z$ over all of $\mathcal{Z}$; hence $c^\top(\mu - \mu^\star) \leq g(\mu)$. Combining these inequalities yields

$$0 \leq F_\theta(\mu) - F_\theta(\mu^\star) \leq g(\mu).$$

The same integer solve therefore finds a direction and bounds the remaining objective error: the oracle supplies both a new column and a global upper bound on optimization error. A small restricted-master gradient or a small change between successive objective values does not establish this bound: undiscovered payoff vectors, the missing columns, may still matter.

9.5 The basic column-generation loop

At a fixed state and with interior candidates, the core iteration is:

1. Solve the restricted master over the active payoff vectors.
2. Compute $c = \nabla F_\theta(\mu)$ and solve the linear oracle over $\mathcal{Z}$.
3. Evaluate $g(\mu) = c^\top(\mu - z^{\text{new}})$.
4. If further improvement is needed, add the outcome and reoptimize the weights.

The gap certifies objective error. A market also needs the safe-update test in [Chapter 10](#), finite-derivative construction in [Chapter 12](#), and settlement rules in [Chapter 13](#). The full source Algorithm 2 appears in the optional concordance, [Section 9.11](#), for reading after those chapters. The basic loop alone is not a rule for committing price resets. For categorical blocks, the same gap argument uses the positive-coordinate entropy extension defined next.

9.6 Categorical markets and the entropy extension

For categorical markets, let

$$c = \nabla \bar{R}(\mu) - \theta.$$

This is the gradient of the smooth expression $\bar{R}(\mu) + C(\theta) - \theta^\top \mu$, which agrees with the correction objective on the coherent price set. That smooth expression is used for derivatives. The extended-valued convex conjugate itself has no ordinary full-space gradient on a simplex. For the reduced binary model at interior prices, $c = \nabla F_\theta(\mu)$ directly, as above.

For categorical LMSR, use the differentiable entropy extension on positive coordinates:

$$\bar{R}(\mu) = b \sum_{j,x} \mu_{j,x} \log \mu_{j,x}, \quad \nabla \bar{R}(\mu)_{j,x} = b(1 + \log \mu_{j,x}).$$

For reduced binary LMSR, $\nabla R(\mu)_i = b \log(\mu_i / (1 - \mu_i))$, the derivative used above. The normalization constraints are implicit in the feasible set. One must not differentiate the indicator of a simplex as though it were an ordinary finite-valued function.

The source’s differentiable-extension discussion must be read on this positive-coordinate domain [P1, §4.1 and Appendix C]. A simplex indicator prevents an ordinary full-space gradient; entropy endpoints can also lack finite subgradients. For binary entropy at zero, the supporting slope would have to lie below every right-hand secant slope, which tends to $-\infty$. Finite entropy values therefore do not justify finite endpoint subgradients. This qualifies the source’s broad “only subdifferentiable” wording.

9.7 The exact correspondence to column generation

For an exactly solved, uncontracted restricted master, let $\mu = \sum_j \lambda_j z_j$ and $c = \nabla F_\theta(\mu)$. The derivative with respect to λ_j is $c^\top z_j$. The KKT conditions give a multiplier τ satisfying

$$c^\top z_j \geq \tau, \quad c^\top z_j = \tau \quad \text{if } \lambda_j > 0.$$

Multiplying by the weights and summing gives $\tau = c^\top \mu$. Thus a new outcome has reduced cost $c^\top z - c^\top \mu$, whose minimum is $-g(\mu)$. A negative reduced cost identifies an improving missing outcome; zero global gap certifies optimality. For contracted atoms, apply this calculation to $(1 - \varepsilon)z + \varepsilon u$. Its gap is different from the original-domain gap needed for economic certification.

Frank–Wolfe object	Decomposition interpretation
Valid payoff vector z	A complete feasible outcome column
Active set $\mathcal{Z}_t$	Columns in the restricted master
Convex weights λ	Master variables mixing outcomes
$F_\theta(\mu)$	Convex nonlinear master objective
$c^\top z - c^\top \mu$	Reduced cost under the conditions above
Integer minimization oracle	Pricing subproblem
$g(\mu)$	Global optimality certificate

The analogy is useful for warm starts, column pools, and restricted-master design. It does not make this a linear master problem or a Benders method: the main generated objects here are feasible outcome columns, not Benders cuts in a separate complicating-variable master.

9.8 Scope of the smooth convergence rate

For the exact-oracle smooth variants analyzed by Jaggi, finite curvature C_F gives objective error at iterate $k \geq 1$ at most $2C_F/(k+2)$. After $K \geq 2$ iterations, at least one iterate among $1, \dots, K$ has gap at most $27C_F/[4(K+2)]$. If the gradient is L -Lipschitz on the fixed compact domain, then $C_F \leq L \operatorname{diam}(\mathcal{M})^2$ [P5, Theorems 1–2 and Section 3]. The diameter enters quadratically through this curvature bound. The gap estimate applies to at least one of the first K iterates; it is not an unconditional bound for the last iterate. Uncontracted entropy can have unbounded curvature, so this theorem alone does not analyze the changing contractions in Chapter 12.

9.9 Sparsity, uniqueness, and what the mixture means

The number of active columns grows by at most one per standard iteration, before pruning or adding multiple oracle solutions. Carathéodory’s theorem says a point in a polytope of affine dimension r can be represented using at most $r+1$ vertices. That is an existence result; an implementation does not automatically keep an active set that small.

Strict convexity of entropy on the relevant positive-coordinate domain gives uniqueness of the optimal candidate price vector. The weights representing that vector need not be unique.

Caution. The active mixture is a witness that the traded marginals are jointly feasible. It is not automatically the market’s uniquely identified posterior over complete outcomes. For example, if only two singleton bits are traded, half the mass on each of 00 and 11 gives the same singleton prices $(1/2, 1/2)$ as half the mass on each of 01 and 10. The untraded conjunction has probability $1/2$ in the first mixture and 0 in the second. Different mixtures with the same traded marginals can assign different probabilities to an untraded conjunction. Do not price new compound events from one arbitrary active mixture without making and documenting an additional modeling choice.

9.10 Optional comparison with a full joint LMSR

A full joint LMSR would involve a partition function such as

$$C_{\text{joint}}(\theta) = b \log \sum_{\omega \in \Omega} \exp(\theta^\top \phi(\omega)/b).$$

Its conjugate has an entropy interpretation over complete outcomes subject to matching marginals. The paper instead starts with independently normalized market blocks; its convex conjugate sums marginal entropy terms. Restricting those marginals to $\mathcal{M}$ does not turn that convex conjugate into joint entropy (Chapter 5 draws the same distinction for the correction objective).

Derivation. Linear optimization over feasible outcomes resembles maximum a posteriori (MAP) inference or combinatorial optimization. Summing exponentials over all outcomes resembles computing a partition function. An oracle for the former does not automatically provide the latter. This distinction explains why an IP-based approximation method is compatible with the complexity results motivating the paper [P1, §§1–2; P7].

9.11 Optional source algorithm concordance

Paper. The paper’s version of the method, PROJECTFW, is based on the fully corrective variant of the Frank–Wolfe algorithm [P5], also known as the simplicial decomposition method. The set of active vertices grows by only one vertex per iteration and is initialized with only a small number of vertices, so the restricted problem, a convex optimization problem of dimension $|\mathcal{Z}_t|$, can be solved efficiently by standard algorithms. The linear optimization over $\mathcal{Z}$ is the computationally expensive step; in all of the paper’s experiments its running time substantially dominated that of the restricted problem [P1, §4.1]. Algorithm 2 restates the algorithm from p. 12 of the paper in the paper’s symbols.  §4.1, Alg. 2, p. 12

Caution: The source output cases are incomplete. The reproduced stopping and return lines permit a fourth case: an uninterrupted, changed, safe return after the small-objective trigger $F(\mu_t) \leq \varepsilon_D$, without the requested extraction guarantee. The trigger bounds the optimal arbitrage value by ε_D ; the separate test $g(\mu_{t*}) \leq F(\mu_{t*})$ certifies nonnegative profit for the returned state. Neither condition implies the fraction guarantee in output case 1, and the state need not remain unchanged as in case 2. The example below exhibits this behavior. The source algorithm is preserved here; an implementation can enforce case 2 by returning the input state on the small-value branch, as the hardened implementation in Chapter 16 does. Record the stopping reason separately from the return action and its profit guarantee (Section 10.4).

Algorithm 2 PROJECTFW: Bregman projection via adaptive fully corrective Frank–Wolfe, restated from Algorithm 2 of the paper (p. 12).

Input: cost function C , state θ , partial outcome σ ,
 IP constraints specified by $\mathbf{A}, \mathbf{b}$,
 approx. ratio $\alpha \in (0, 1)$, initial contraction $\varepsilon_0 \in (0, 1)$, convergence threshold $\varepsilon_D > 0$

Output: extended partial outcome $\hat{\sigma} \supseteq \sigma$
 state $\hat{\theta}$, whose price vector is an approx. Bregman projection of θ on $\mathcal{M}_{\hat{\sigma}}$ in the sense that one of the following holds:

1. $p_{\hat{\sigma}}(\hat{\theta}) \in \mathcal{M}_{\hat{\sigma}}$ and moving from θ to $\hat{\theta}$ guarantees the profit of $\alpha D_{\hat{\sigma}}(\mu^* \parallel \theta)$
2. $\hat{\theta} = \theta$ and $D_{\hat{\sigma}}(\mu^* \parallel \theta) \leq \varepsilon_D$
3. algorithm was interrupted; moving from θ to $\hat{\theta}$ guarantees a non-negative profit where $\mu^* = \arg \min_{\mu \in \mathcal{M}_{\hat{\sigma}}} D_{\hat{\sigma}}(\mu \parallel \theta)$

Initialize the interior point, active vertex set, and extend the partial outcome:
 $(u, \mathcal{Z}_0, \hat{\sigma}) \leftarrow \text{INITFW}(\sigma, \mathbf{A}, \mathbf{b})$

Define the objective function:
 $F(\mu) := \bar{R}_{\hat{\sigma}}(\mu) - \theta \cdot \mu + C_{\hat{\sigma}}(\theta)$

for $t = 1, 2, \dots$ **do**
 perform a FW iteration on the contracted polytope:
 let $\mathcal{Z}' = (1 - \varepsilon_{t-1})\mathcal{Z}_{t-1} + \varepsilon_{t-1}u$ denote the contracted active set
 $\mu_t \leftarrow \arg \min_{\mu \in \text{conv}(\mathcal{Z}')} F(\mu)$
 $\theta_t \leftarrow \nabla \bar{R}_{\hat{\sigma}}(\mu_t)$
 call IP solver to find the descent vertex (note that $\nabla F(\mu_t) = \theta_t - \theta$):
 $z_t \leftarrow \arg \min_{z \in \mathcal{Z}_{\hat{\sigma}}} (\theta_t - \theta) \cdot z$
 $\mathcal{Z}_t = \mathcal{Z}_{t-1} \cup \{z_t\}$
 compute the FW gap $g(\mu_t) = (\theta_t - \theta) \cdot (\mu_t - z_t)$
 update the best-iterate-so-far $t^* \leftarrow \arg \max_{\tau \leq t} [F(\mu_\tau) - g(\mu_\tau)]$
 check stopping conditions:
 if $g(\mu_t) \leq (1 - \alpha)F(\mu_t)$, or $F(\mu_t) \leq \varepsilon_D$, or termination requested **then**
 return $\hat{\sigma}$ and $\hat{\theta} = \begin{cases} \theta_{t^*} & \text{if } g(\mu_{t^*}) \leq F(\mu_{t^*}) \\ \theta & \text{otherwise} \end{cases}$
 end if
 adapt contraction if necessary:
 let $g_u = (\theta_t - \theta) \cdot (\mu_t - u)$
 if $g_u < 0$ and $g(\mu_t)/(-4g_u) < \varepsilon_{t-1}$ **then**
 $\varepsilon_t \leftarrow \min\{g(\mu_t)/(-4g_u), \varepsilon_{t-1}/2\}$
 else
 $\varepsilon_t \leftarrow \varepsilon_{t-1}$
 end if
end for

The source objective F is F_θ with the settled cost and entropy extension, and D denotes the same correction value. Its θ_t is the candidate inverse state, $\mathcal{Z}_t$ is the active set, and ε_D is this tutorial's ε_F . The stopping line uses the same index t in $g(\mu_t)$ and $F(\mu_t)$.

A small-value return without the extraction guarantee. Derivation. Use the six-security triangle of Section 8.3, with $b = 1$, $\alpha = 0.9$, $\varepsilon_0 = 0.1$, $\varepsilon_D = 1$, and

$$\theta = (-4/5, -4/5, 1/5, 9/5, -1/5, -1/5).$$

The all-zero and all-one outcomes are permitted tied witnesses for INITFW. Their anchor is $u = \frac{1}{2}\mathbf{1}$. Since $\sum_i \theta_i = 0$, the first contracted master has minimizer $\mu = \frac{1}{2}\mathbf{1}$, with inverse state zero. Direct evaluation of the eight outcomes gives

$$\begin{aligned} F_\theta(\mu) &= C(\theta) - 6 \log 2 \approx 0.530712, \\ g(\mu) &= 1/5, \quad \Pi = F_\theta(\mu) - g(\mu) \approx 0.330712. \end{aligned}$$

The small-value trigger succeeds and the extraction test fails. The return rule nevertheless selects the changed state zero because $\Pi > 0$.

To compare this profit with the true optimum, set $s = (1 + e^{-1/15})^{-1}$ and $q = (1 + e^{-1/5})^{-1}$. The candidate $\mu^\star = (s, s, q, s, 1 - q, 1 - q)$ is coherent: weights $(2 - s - 2q, 2q - 1, s + q - 1, 1 - q)$ on worlds 000, 001, 110, 111 are positive, sum to one, and yield those marginals. Its inverse-state trade is $c^\star = (13/15, 13/15, 0, -26/15, 0, 0)$. For every outcome, $(c^\star)^\top z = (13/15)(x_1 + x_2 - 2x_1x_2) \geq 0$, with equality on the mixture's support. Its gap is therefore zero, proving optimality. Thus

$$F_\theta(\mu^\star) \approx 0.514071, \quad \alpha F_\theta(\mu^\star) \approx 0.462664 > \Pi.$$

The return is safe but satisfies none of the three stated output cases. Scaling liquidity, state, and ε_D by the same positive constant makes the absolute threshold arbitrarily small while preserving this failure.

Exercises.

1. Repeat the two-contract oracle calculation at $\mu = (0.6, 0.6)$ and compute the gap. Which outcome enters, and how does its reduced cost compare with the gap when the restricted master is solved exactly?
2. Verify that the two distributions on 00/11 and 01/10 have identical singleton marginals and different conjunction probabilities. Explain why adding the conjunction as a traded coordinate changes the comparison.
3. Identify every additional ingredient used by Algorithm 2 beyond the basic loop, and locate its definition in Chapters 10–13.

Chapter 10

Safe updates and profit certificates

Derive a test for accepting a candidate state and distinguish safe updates, extraction targets, and unchanged returns.

10.1 Converting a candidate price vector back to a state

For a coherent candidate μ , require $0 < \mu_i < 1$ in reduced binary coordinates. For categorical blocks require positive probabilities on surviving categories, after settled coordinates have been handled. Under these domain conditions, set

$$\widehat{\theta} = \nabla \bar{R}(\mu).$$

For the paper's entropy construction, conjugacy gives $p(\widehat{\theta}) = \mu$. The prospective internal trade is

$$\Delta = \widehat{\theta} - \theta.$$

For the binary LMSR, where R is differentiable at interior candidates, the pricing state that quotes μ is

$$\widehat{\theta} = \nabla R(\mu), \quad \widehat{\theta}_i = b \log \frac{\mu_i}{1 - \mu_i}.$$

Consequently $c = \widehat{\theta} - \theta$, the oracle direction of [Equation \(9.1\)](#), is also the *candidate trade*: $\Delta = c$.

For categorical LMSR, $\widehat{\theta}_{j,x} = b \log \mu_{j,x}$ gives the same prices as $b(1 + \log \mu_{j,x})$; the blockwise constant cancels from the trade-profit calculation. A complete categorical payoff block sums to one, so adding $k\mathbf{1}$ adds k to both its trade charge and settlement payoff; see the cash-equivalence calculation in [Chapter 2](#).

10.2 Why the gap determines safe profit

Define the worst-case profit of the corresponding hypothetical trader:

$$\Pi(\Delta; \theta) = \min_{z \in \mathcal{Z}} [\Delta^\top z - C(\theta + \Delta) + C(\theta)].$$

Under any distribution with expected payoff μ , the candidate trade's expected settlement payoff is $c^\top \mu$. Its worst settlement payoff is $\min_z c^\top z$. The Frank–Wolfe gap $g(\mu)$ of [Equation \(9.2\)](#) is precisely the difference between those two amounts.

The cost of the candidate trade is $C(\hat{\theta}) - C(\theta)$. Conjugate equality gives

$$R(\mu) = \hat{\theta}^\top \mu - C(\hat{\theta}),$$

so

$$F_\theta(\mu) = c^\top \mu - [C(\hat{\theta}) - C(\theta)].$$

Thus $F_\theta(\mu)$ equals the candidate trade's expected profit under a distribution producing μ . Subtracting the expected-payoff-minus-worst-payoff difference gives its *worst-case profit*:

$$\Pi(c; \theta) = \min_{z \in \mathcal{Z}} c^\top z - [C(\hat{\theta}) - C(\theta)] = F_\theta(\mu) - g(\mu). \quad (10.1)$$

Paper. Proposition 4.1 of the paper shows the update is safe when $F_\theta(\mu) - g(\mu) \geq 0$. In the stated conjugate-gradient construction, Appendix D of the paper actually derives the identity (10.1) [P1, §4.2 and App. D]. The proposition states a lower bound; its proof gives equality for this construction. The optional source proof is in Section 10.6.

■ §4.2, Prop. 4.1

This identity is one of the most useful implementation checks in the entire paper. On a small enumerated instance, compute the left side directly over every outcome and compare it with the right side.

A coherent candidate can still have negative worst-case profit (Section 10.3). Acceptance requires $F_\theta(\mu) - g(\mu) \geq 0$.

For an internal correction, this hypothetical trade certifies the update. The actual customer positions and cash account do not change. Their accounting is treated separately in Section 14.2.

10.3 Why a feasible price reset can still be wrong

Suppose a solver returns any coherent μ . Its output prices would have no arbitrage, but it may be far from the candidate minimizing the correction objective. The state change can have $F_\theta(\mu) - g < 0$, meaning that in at least one outcome the corresponding hypothetical trade loses money.

A negative value fails the sufficient nonnegative-profit condition used to preserve the original loss bound. The reset changes neither existing customer holdings nor cash, so this failure is not an immediate cash loss or proof that the original budget has already been exceeded.

For two identical contracts, take current prices (0.8, 0.4) and $b = 1$. Every target (r, r) in the following table is coherent. Use its finite binary inverse state to construct the candidate trade.

r	$F_\theta(\mu)$	$g(\mu)$	Π	Interpretation
0.500000	0.243555	0.490415	-0.246860	Fails the safety test
0.600000	0.185743	0.067960	0.117783	Safe, suboptimal extraction
0.620204	0.184021	0	0.184021	Optimal extraction

For $r = 0.5$, the candidate inverse state is (0, 0), the expected payoff of its trade is -0.490415, the worst payoff is -0.980829, and the common trade charge is -0.733969. Subtracting that charge gives expected profit 0.243555 and worst profit -0.246860. The positive objective alone is therefore insufficient. The optimal row uses $r = \sqrt{8/3}/(1 + \sqrt{8/3})$ before rounding. The triangle example in Chapter 15 provides a larger cross-check with worst profit approximately -0.709166 at $b = 1$.

10.4 The paper's stopping conditions

Choose a desired *extraction fraction* $\alpha \in (0, 1)$ and a small absolute threshold $\varepsilon_F > 0$ [P1, §4.2, Alg. 2 Algorithm 2 and §4.2]. (The paper writes ε_D for this threshold; see Algorithm 2.)

Extraction target met. Stop when

$$g(\mu) \leq (1 - \alpha)F_\theta(\mu). \quad (10.2)$$

Then

$$\Pi = F_\theta(\mu) - g \geq \alpha F_\theta(\mu) \geq \alpha F_\theta(\mu^*).$$

The update captures at least an α fraction of the maximum available guaranteed arbitrage profit. The main paper identifies $F_\theta(\mu^*)$ with that maximum under its cost-function assumptions [P1, Proposition 2.4 and App. D]; see Equation (6.1).

Unchanged with a small arbitrage bound. If

$$F_\theta(\mu) \leq \varepsilon_F,$$

then $F_\theta(\mu^*) \leq \varepsilon_F$. Keeping the original state is permitted; it might still have a small arbitrage opportunity. The threshold concerns the maximum guaranteed profit, not a coordinatewise distance to the coherent set. The source algorithm can instead return a changed safe state on this trigger, without meeting the extraction target; the qualification in Section 9.11 explains this additional return case.

Interruption. Track the best certified value of $F_\theta(\mu) - g$ across all iterates. If a nonnegative one exists, return its corresponding state. Otherwise keep the previous state. This is why stopping after a fixed number of iterations can preserve the loss bound even when it fails to complete the requested projection.

A return records both the action taken and the stopping reason:

Status	Action and evidence
Extraction target met	Accept the candidate with the relative guarantee.
Safe update before target	Accept a coherent candidate with $\Pi \geq 0$, for example on interruption or the source's small-value trigger.
Unchanged with small arbitrage bound	Keep θ because a coherent candidate proves $F_\theta(\mu^*) \leq \varepsilon_F$.
Unchanged without a certificate	Keep θ when the computation has not established an applicable guarantee.

Coherence of an accepted update. An accepted coherent update removes arbitrage at the returned prices even when $\alpha < 1$. The extraction fraction controls how much guaranteed internal profit the correction captures. Arbitrage may remain when the algorithm returns the unchanged state, or reappear after subsequent real trades.

10.5 Do not drop the constant from the stopping rule

For optimization, minimizing $R(\mu) - \theta^\top \mu$ is equivalent to minimizing $F_\theta(\mu)$. For the economic stopping condition, they are not interchangeable:

$$F_\theta(\mu) = R(\mu) - \theta^\top \mu + C(\theta).$$

The $C(\theta)$ term must be included when evaluating $F_\theta(\mu) - g$, the relative extraction rule (10.2), or the absolute arbitrage-value threshold. A solver may optimize an objective without the constant, but the surrounding market engine must add it back.

A normalization check makes this requirement concrete. Replace C by $C + k$ and R by $R - k$. Prices and trade charges are unchanged; the two constants cancel in F_θ , and the gap and every acceptance decision remain unchanged. Test this invariance when separating an optimizer's objective from the reported economic value.

10.6 Optional source proof concordance

Proposition 4.1 states a guaranteed lower bound; Appendix D derives equality for the prescribed construction [P1, Proposition 4.1 and Appendix D]. The following source proof is optional after the three-amount derivation above.

Proposition 4.1 of the paper (proof in Appendix D). *Consider a purchase that moves the market from a state θ to a new state $\hat{\theta} = \nabla \bar{R}(\hat{\mu})$. The resulting profit is guaranteed to be at least $D(\hat{\mu} \parallel \theta) - g(\hat{\mu})$.*

Proof, following Appendix D of the paper. The guaranteed profit when moving from θ to $\hat{\theta}$ is

$$\begin{aligned} & \min_{\omega \in \Omega} [(\hat{\theta} - \theta) \cdot \phi(\omega) - C(\hat{\theta}) + C(\theta)] \\ &= \min_{\mu \in \mathcal{M}} [(\hat{\theta} - \theta) \cdot \mu - C(\hat{\theta}) + C(\theta)] \end{aligned} \quad (16)$$

$$\begin{aligned} &= \min_{\mu \in \mathcal{M}} [(\hat{\theta} - \theta) \cdot (\mu - \hat{\mu}) + \hat{\theta} \cdot \hat{\mu} - C(\hat{\theta}) - \theta \cdot \hat{\mu} + C(\theta)] \\ &= \min_{\mu \in \mathcal{M}} [(\hat{\theta} - \theta) \cdot (\mu - \hat{\mu}) + R(\hat{\mu}) - \theta \cdot \hat{\mu} + C(\theta)] \end{aligned} \quad (17)$$

$$= D(\hat{\mu} \parallel \theta) - g(\hat{\mu}). \quad (18)$$

Eq. (16) follows because the minimized objective is linear in $\phi(\omega)$. Eq. (17) follows from the definition of R . Finally, Eq. (18) follows because $\nabla F(\hat{\mu}) = \nabla \bar{R}(\hat{\mu}) - \theta = \hat{\theta} - \theta$, and hence

$$g(\hat{\mu}) = \max_{\mu \in \mathcal{M}} [(\hat{\theta} - \theta) \cdot (\hat{\mu} - \mu)]. \quad \square$$

The equation numbers (16)–(18) are the paper's. Here its D is the correction objective F_θ ; the proof applies to the prescribed entropy-gradient construction, not arbitrary choices of inverse-price state.

Exercises.

1. Recompute the table from the binary cost and the two possible outcomes. For $\alpha = 0.5$ and 0.9 , classify each row by both action and stopping reason.
2. Add $k = 7$ to the cost and subtract 7 from its conjugate. Verify that the charges, prices, correction objective, gap, and acceptance decisions agree.
3. An interrupted run has found a safe candidate but missed its extraction target. State what its returned prices certify and what the run failed to achieve.

Chapter 11

Certificates from time-limited solves

Use time-limited solver bounds safely, express tolerances in economic units, and prevent stale-state certificates from being committed.

Status: Implementation extension. The paper’s oracle is solved to its specified optimum. This chapter derives certificates from a feasible incumbent and a valid lower bound when a solve stops early. These are implementation extensions, not claims about the original experiments. Solver fields follow the versioned documentation [S1; S2]; numerical audits refer to the starter experiment in [Chapter 16](#).

11.1 Separate the incumbent from the objective bound

Derivation and implementation extension. For the oracle objective $c^\top z$ of [Equation \(9.1\)](#), let

$$m^\star = \min_{z \in \mathcal{Z}} c^\top z.$$

A time-limited minimization solve may return a valid lower bound m_{lb} and an integer *incumbent*, a feasible payoff vector with objective value m_{inc} , so

$$m_{\text{lb}} \leq m^\star \leq m_{\text{inc}}.$$

Subtracting from $c^\top \mu$ reverses the inequalities:

$$\underbrace{c^\top \mu - m_{\text{inc}}}_{\text{lower estimate}} \leq g(\mu) \leq \underbrace{c^\top \mu - m_{\text{lb}}}_{\bar{g}, \text{ upper bound}}.$$

Define the upper bound

$$\bar{g} = c^\top \mu - m_{\text{lb}}. \tag{11.1}$$

A feasible incumbent gives a lower bound on the true Frank–Wolfe gap. Acceptance needs an upper bound because that gap is subtracted from $F_\theta(\mu)$. A poor incumbent can make the apparent gap look small or even negative despite substantial missing improvement.

Use the global lower objective bound instead. Since $F_\theta(\mu) = c^\top \mu - [C(\hat{\theta}) - C(\theta)]$ ([Section 10.2](#)), the certified lower profit bound is

$$\Pi \geq F_\theta(\mu) - \bar{g} = m_{\text{lb}} - [C(\hat{\theta}) - C(\theta)]. \tag{11.2}$$

This can certify a safe update before the MIP closes its optimality gap. A heuristic feasible outcome can still be added as a column, but it cannot by itself certify that no more dangerous outcome exists. These fields have independent roles: a validated incumbent enlarges the active set, while a valid bound may certify an already available coherent candidate even if the current solve supplies no new incumbent. A state update still requires that candidate and its prescribed finite inverse state.

For illustration, suppose $F_\theta(\mu) = 1.5$, $c^\top \mu = 10$, $m_{lb} = 7$, and $m_{inc} = 9$. Then $1 \leq g \leq 3$. Using only the feasible payoff vector suggests profit $1.5 - 1 = 0.5$, whereas the valid lower profit bound is $1.5 - 3 = -1.5$. Thus $\Pi \in [-1.5, 0.5]$: the available evidence does not certify a safe update, but it also does not prove the update unsafe.

Use $\bar{g}$ in the acceptance tests of [Sections 10.2](#) and [10.4](#). Check solver tolerances, candidate feasibility, and the frozen state before accepting. A materially negative computed $\bar{g}$ indicates inconsistent inputs or an invalid bound; replacing it by zero does not repair the certificate.

SciPy 1.17.0 (the version used for the historical starter results) exposes `mip_lp`, which reports the minimization bound as `mip_dual_bound`; Gurobi exposes the bound as `ObjBound` and the incumbent objective as `ObjVal`. A field can be unavailable at a timeout; check it independently of solver status. Their relative MIP optimality gaps are not the outer Frank–Wolfe gap [[S1](#); [S2](#)].

Before using a reported bound, verify a finite value, minimization sense, objective scaling, matching model/state version, and numerical status. If the solver used coefficients c/s with $s > 0$, multiply its bound by s to recover the original objective units. A missing bound prevents bound-based certification but need not invalidate a separately checked incumbent. Validate any returned assignment against the payoff constraints before adding it. Neither a success flag nor rounding an incumbent establishes this contract. Record the solver/library version and its tolerances with the run; the starter results use SciPy 1.17.0 [[S1](#)]. There is also an absolute stopping test ([Section 10.4](#)). If a coherent candidate satisfies $F_\theta(\mu) \leq \varepsilon_F$ for a chosen tolerance $\varepsilon_F > 0$, then $0 \leq F_\theta(\mu^\star) \leq \varepsilon_F$. Keeping the original state leaves at most that much available guaranteed arbitrage profit. If time runs out before either a safe correction or this small-value condition is established, keep the original state and report that computation was insufficient.

11.2 A useful action table

For every update row below, assume a coherent candidate, its prescribed finite inverse state, reliable cost/objective evaluation, and a matching frozen candidate, state, cost, payoff model, and version. Numerical error must be handled separately; the table's bounds are mathematical bounds.

Available evidence	Permitted algorithm action
Valid new integer payoff vector, no useful bound	Add the column and continue the restricted master.
Valid bound proving $F_\theta(\mu) - \bar{g} \geq 0$	A coherent candidate can be accepted as a safe internal correction.
Valid bound proving $\bar{g} \leq (1 - \alpha)F_\theta(\mu)$	Accept with the requested fraction-of-optimal-profit guarantee.
Feasible coherent candidate with $F_\theta(\mu) \leq \varepsilon_F$	Keeping the original state leaves at most ε_F optimal arbitrage value.
No valid certificate, deadline reached	Keep the old pricing state; retain reusable computation if appropriate.
Proven infeasible outcome model	Stop this correction and resolve the model/settlement inconsistency.

All certificates must refer to the same outcome set, cost function, and frozen original state. A negative upper gap beyond roundoff is a diagnostic failure: either the candidate is infeasible, the bound is not valid for that model, or some state/objective data were mixed.

11.3 Absolute economic tolerances

MIP objective coefficients change scale with b and with the current gradient. A solver's relative objective tolerance is therefore not automatically a useful tolerance in dollars of guaranteed profit.

Record at least the incumbent value, global bound, objective scaling, feasibility tolerances, integrality tolerances, and the resulting absolute profit lower bound. Set a positive acceptance margin in the market's monetary units. Check reconstructed rounded payoff vectors against the actual logical constraints before adding them to the active set.

Status: Numerical finding from the starter experiment. A very stringent test in the starter experiment (Chapter 16) exposed an objective discrepancy on the order of 10^{-7} despite the solver reporting an optimal status. The code normalizes the objective, subtracts an explicit chosen numerical allowance from the returned lower bound, and then audits the resulting guarded solver estimate against full enumeration. This is evidence for the tiny tested instance; the allowance is not a general exact-arithmetic proof for arbitrary models.

For larger experiments, distinguish a floating-point solver certificate under specified tolerances from a formally verified bound. Do not turn numerical rounding into a theorem by labeling an approximate solver result exact.

The starter script clamps a negative computed upper gap to zero and subsequently audits the bound against full enumeration. A general implementation should reject a materially

negative upper gap before acceptance. The later enumeration audit supports the tiny example; the clamp alone does not validate a bound. A required interface test injects an alleged lower bound $m_{lb} > c^\top \mu + \eta_{eval}$, where η_{eval} is the allowed evaluation error. The interface must reject or mark this certificate invalid before any acceptance test; applying $\max(0, \bar{g})$ is not a repair. This is a hardening requirement, while the archived starter snapshot remains unchanged.

11.4 A safe online integration protocol

Implementation recommendation: Frozen snapshots and atomic commits. Assign a version to the market state and settlement model. A correction job reads a frozen snapshot containing θ , the cost specification, the outcome constraints, and the version.

On completion, atomically check that the snapshot is still current before committing a correction. If a new trade or settlement arrived, either discard/restart the candidate or recertify it against the new state. Old outcome columns may be reused only after checking the new settlement constraints. To recertify a candidate, first check its feasibility in the new outcome set. Then recompute its inverse-state trade, cost difference, correction objective, and oracle coefficients, and obtain a bound for that objective. A bound from the old objective cannot certify the new trade.

For a first implementation, serialize real trades and corrections. Add asynchronous execution only after the accounting and certification tests pass. Log whether each correction completed, returned a safe partial update, returned unchanged because arbitrage value was tiny, or returned unchanged because computation was insufficient. Those are different outcomes.

Exercises.

1. A timeout returns $F_\theta(\mu) = 1.5$, $c^\top \mu = 10$, a feasible incumbent of value 9, and a valid lower bound of 7. Compute the profit interval and permitted actions. Repeat with the incumbent missing, then with the bound missing, assuming a coherent candidate was already available.
2. A real trade arrives after a correction snapshot but before commit. List the values to recompute and the columns that can be reused. Repeat when new settlement information also changes the payoff set.
3. Inject a lower bound above $c^\top \mu$ by more than the evaluation allowance. Explain why an interface must reject it before clamping or acceptance.

Chapter 12

Entropy boundaries, contraction, and the interior point

Keep entropy derivatives finite while constructing coherent candidates, and distinguish the contracted optimization gap from the original-domain profit certificate.

12.1 Why finite states cannot quote binary endpoints

For one unresolved binary LMSR coordinate,

$$R'_i(\mu_i) = b \log \frac{\mu_i}{1 - \mu_i}.$$

The derivative tends to $-\infty$ at zero and $+\infty$ at one. The inverse state therefore cannot be finite at either endpoint. Yet the conjugate value remains finite: $\mu_i \log \mu_i \rightarrow 0$ as $\mu_i \downarrow 0$, and the same continuous convention handles the other entropy terms. Finite endpoint values and infinite endpoint derivatives are compatible.

They also need not give finite endpoint subgradients. At zero, a finite supporting slope for extended binary entropy would have to be below every right-hand secant slope; those slopes tend to $-\infty$. The claim that the whole extended entropy is everywhere subdifferentiable is therefore false. For categorical blocks, removing the simplex indicator permits derivatives of the entropy formula on positive active coordinates. This does not make that formula smooth at zero or differentiate the indicator itself.

12.2 Why ordinary smooth Frank–Wolfe analysis is insufficient

In the fixed-domain binary model, the Hessian is diagonal with entries

$$R''_i(\mu_i) = \frac{b}{\mu_i(1 - \mu_i)}.$$

On $[\eta, 1 - \eta]^d$, with $0 < \eta \leq 1/2$, its Euclidean operator norm is at most $b/[\eta(1 - \eta)]$. This bounds the gradient's Lipschitz constant there, but diverges as the distance η to the box faces tends to zero. For categorical entropy on positive coordinates,

$$\frac{\partial \bar{R}}{\partial \mu_i} = b(1 + \log \mu_i), \quad \frac{\partial^2 \bar{R}}{\partial \mu_i^2} = \frac{b}{\mu_i}.$$

The same problem occurs near zero. A line search or restricted master that hits an endpoint can therefore produce an infinite derivative even though its objective value is finite.

The paper adapts Barrier Frank–Wolfe by restricting optimization to a contracted domain [P1, §4.1 and Appendix C][P6]. Its general controlled-growth condition bounds the gradient Lipschitz constant by $L_\eta \leq L\eta^{-\kappa}$. We use η for coordinate distance, κ for the growth exponent, and ε for contraction. The binary and categorical Hessian bounds have growth exponent $\kappa = 1$ near the relevant faces. The settled-domain source assumptions appear only in the optional concordance, after the notation they require is defined.

12.3 A coordinate-interior anchor

An *anchor* $u \in \mathcal{M}$ must satisfy $0 < u_i < 1$ for every unsettled binary coordinate. Find valid payoff vectors witnessing each possible coordinate value and average them. If both zero and one occur among the witnesses for coordinate i , their average has $0 < u_i < 1$. For a categorical block, every surviving category needs a witness with payoff one, making its average probability positive.

A witness proves possibility. If a certified optimum or infeasibility result rules out a requested value, the coordinate is fixed to the opposite value. A timeout without that certificate leaves it unresolved; it cannot justify settlement or a claim that the anchor has been completed. This is the logic behind the paper’s INITFW routine [P1, §4.3 and Algorithm 3].

Implementation recommendation: Witness averaging in tutorial notation. Start with a validated feasible outcome and a witness set W containing it. Record all pairs (i, z_i) seen in W . For each unseen coordinate/value pair (i, β) , with $\beta \in \{0, 1\}$, search for a valid outcome with $z_i = \beta$. Add a found witness and all of its coordinate/value pairs. If a sound certificate rules out β , record the forced value $1 - \beta$. If the search is inconclusive, leave the initialization incomplete. After all values have been witnessed or ruled out, average the distinct payoff vectors in W to obtain u .

Only the active coordinate values must avoid zero and one. The anchor may still lie on another face of $\mathcal{M}$. For the triangle market, averaging the all-zero and all-one payoff vectors gives $u_i = 1/2$ for every coordinate. It lies on faces such as $y_{12} = x_1$ and those two points do not span the full coherent set. Coordinate interior is sufficient for the entropy gradient argument; full relative interior is unnecessary.

12.4 Keep derivatives finite without changing the certificate

For $0 < \varepsilon < 1$, define the contracted set

$$\mathcal{M}' = (1 - \varepsilon)\mathcal{M} + \varepsilon u. \quad (12.1)$$

Every candidate has the form $\mu = (1 - \varepsilon)v + \varepsilon u$ with $v \in \mathcal{M}$. Since $0 \leq v_i \leq 1$,

$$\varepsilon u_i \leq \mu_i \leq 1 - \varepsilon(1 - u_i).$$

With $\rho = \min_i \{u_i, 1 - u_i\} > 0$ over active coordinates, we obtain $\mu_i \in [\varepsilon\rho, 1 - \varepsilon\rho]$. Thus the binary Hessian norm is at most $b/[\varepsilon\rho(1 - \varepsilon\rho)]$. The coordinate buffer is explicit, and

convexity ensures that the candidate remains coherent. For active vertices $z_1, \dots, z_k$, the restricted master uses

$$\mu = (1 - \varepsilon) \sum_{r=1}^k \lambda_r z_r + \varepsilon u, \quad \lambda_r \geq 0, \quad \sum_r \lambda_r = 1.$$

A fixed positive contraction can exclude the original optimum and prevent a stringent extraction target from being met. Reducing ε allows closer approach while each positive value keeps the derivatives finite.

12.4.1 Original-domain versus contracted-domain gap

For the candidate trade c , compute the economic certificate using the original payoff set:

$$g = c^\top (\mu - z^{\text{new}}), \quad z^{\text{new}} \in \arg \min_{z \in \mathcal{Z}} c^\top z.$$

Define $g_u = c^\top (\mu - u)$; the derivative for moving toward the anchor is $-g_u$. The gap over the contracted set is

$$g_\varepsilon = (1 - \varepsilon)g + \varepsilon g_u.$$

Solving the contracted problem accurately can make this gap zero while the original gap remains positive. Only the original gap belongs in the profit identity $\Pi = F_\theta(\mu) - g$ of Equation (10.1).

For a numerical example, take one binary market with $b = 1$, current price 0.9, anchor 0.5, and $\varepsilon = 0.4$. The contracted interval is $[0.2, 0.8]$, and its objective minimizer is $\mu = 0.8$. There,

$$c = \log(0.8/0.2) - \log(0.9/0.1) = \log(4/9) < 0.$$

The contracted oracle returns 0.8 and has zero gap, but the original oracle returns 1, giving

$$g = c(0.8 - 1) \approx 0.162186, \quad F_\theta(0.8) \approx 0.044403, \quad \Pi \approx -0.117783.$$

Using the contracted gap would report positive profit for a candidate whose actual minimum profit is negative.

With an exact oracle, the paper reduces contraction according to

$$\text{if } g_u < 0 \text{ and } \frac{g}{-4g_u} < \varepsilon, \quad \varepsilon \leftarrow \min \left\{ \frac{g}{-4g_u}, \frac{\varepsilon}{2} \right\}.$$

Otherwise it leaves ε unchanged [P1, Algorithm 2]. A negative g_u means that moving toward the anchor increases the objective; the rule relaxes an overly restrictive contraction when the stated threshold holds. The starter's use of a guarded upper gap in this rule is an engineering modification. The source's convergence proof does not automatically cover it.

Contraction changes the optimization domain without asserting new settlement information. The next chapter changes the market after observed information or a sound logical certificate fixes payoffs.

12.5 Optional source concordance

This section records the general assumptions and source pseudocode for reading after the fixed-domain explanation. In source notation, $\mathcal{I}$ indexes all d security coordinates. A feasible partial outcome σ fixes the coordinates in $\mathcal{I}_\sigma \subseteq \mathcal{I}$. The affine space V_σ fixes those values, $\mathcal{Z}_\sigma = \mathcal{Z} \cap V_\sigma$ is nonempty, and $\mathcal{M}_\sigma = \text{conv}(\mathcal{Z}_\sigma)$. Restricting the conjugate to this affine space gives R_σ ; Section 13.2 develops the cost change and its accounting.

In the following source statements, ε is distance to the unit-box faces and p is a growth exponent. The main text instead uses η and κ , reserving ε for contraction and p for prices. Differentiability is required on active interior coordinates. It does not assert finite entropy subgradients at every boundary point. Example C.3 is written with unit liquidity; multiplying by b scales its Hessian bound. Its convex indicator $\mathbb{I}\{P\}$ equals zero when the condition P holds and $+\infty$ otherwise. This differs from the zero–one event indicator $\mathbf{1}\{P\}$ used for payoffs.

Definition C.1 of the paper. *Let $S \subseteq [0, 1]^n$ be a compact convex set. We say that a convex function F exhibits controlled growth on S if it is differentiable on $S \cap (0, 1)^n$ and if there exists a fixed $p \geq 0$ and $L \geq 0$ such that for any $\varepsilon > 0$, the gradient ∇F has a bounded Lipschitz constant $L_\varepsilon \leq L\varepsilon^{-p}$ over $S \cap [\varepsilon, 1 - \varepsilon]^n$.*

Assumption C.2 of the paper. *R has a convex extension $\bar{R}$ such that for all partial outcomes σ , when $\bar{R}$ is viewed as a function on V_σ , it exhibits controlled growth on $\mathcal{M}_\sigma$.*

Example C.3 of the paper (Controlled growth for LMSR). *We define the extension of negative entropy over the non-negative orthant, $\bar{R}(\mu) = \mathbb{I}\{\mu \geq \mathbf{0}\} + \sum_{i \in \mathcal{I}} \mu_i \log \mu_i$, which yields $\bar{R}_\sigma(\mu) = \mathbb{I}\{\mu_i \geq 0 \text{ for all } i \notin \mathcal{I}_\sigma\} + \sum_{i \notin \mathcal{I}_\sigma} \mu_i \log \mu_i$. The Hessian is a diagonal matrix with entries $1/\mu_i$, so its operator norm is $\max_{i \notin \mathcal{I}_\sigma} 1/\mu_i$, and thus $L_\varepsilon = O(1/\varepsilon)$, which satisfies the controlled growth condition with $p = 1$.*

The gradient of the restricted entropy uses the unsettled coordinates; the source inserts zeros in settled positions to regard it as a full vector. On the active interior this gradient is a subgradient of the restricted conjugate and gives the prescribed inverse state [P1, Appendix C].

The original INTFW pseudocode below assumes exact oracle conclusions. Its Boolean b is unrelated to liquidity, and its record C is unrelated to the cost function. The main construction above uses β and the witness record to avoid these collisions. A failed timed search must never take the source algorithm’s settlement branch.

Algorithm 3 INITFW. Initialization for PROJECTFW (Algorithm 3 of the paper, p. 13).

Input: partial outcome σ , IP constraints specified by $\mathbf{A}$, $\mathbf{b}$
Output: extended partial outcome $\hat{\sigma} \supseteq \sigma$
 point $\mathbf{u} \in \mathcal{M}_{\hat{\sigma}}$ such that $u_i \in (0, 1)$ for $i \notin \mathcal{I}_{\hat{\sigma}}$
 non-empty set $\mathcal{Z}_0$ of vertices of $\mathcal{M}_{\hat{\sigma}}$

 Initialize $\mathcal{Z}_0 \leftarrow \emptyset$, $\hat{\sigma} \leftarrow \sigma$, $C \leftarrow \emptyset$
for each $i \in \mathcal{I} \setminus \mathcal{I}_{\sigma}$ and each $b \in \{0, 1\}$ **do**

 if $(i, b) \notin C$ **then**

 call IP solver to find $\hat{\mathbf{z}} = \arg \max_{\mathbf{z} \in \mathcal{Z}_{\sigma}} (2b - 1)z_i$
if $\hat{z}_i = b$ **then**
 $\mathcal{Z}_0 \leftarrow \mathcal{Z}_0 \cup \{\hat{\mathbf{z}}\}$
 $C \leftarrow C \cup \{(j, \hat{z}_j) : j \in \mathcal{I}\}$
else
 $\hat{\sigma} \leftarrow \hat{\sigma} \cup \{(i, 1 - b)\}$
end if
end if
end for
if $\mathcal{Z}_0 = \emptyset$ **then**
 $\mathcal{Z}_0 \leftarrow \{\text{the unique point compatible with } \hat{\sigma}\}$
end if
return $\hat{\sigma}$, $\mathcal{Z}_0$, and $\mathbf{u} = \frac{1}{|\mathcal{Z}_0|} \sum_{\mathbf{z} \in \mathcal{Z}_0} \mathbf{z}$

Exercises.

1. Use the triangle's all-zero and all-one outcomes to construct the anchor. For $\varepsilon = 0.2$, calculate the coordinate buffer and binary Hessian bound. Identify a non-box face that still contains the anchor.
2. Recompute both gaps and the profit in the one-dimensional example. Reduce contraction to 0.1 and locate the new contracted optimum.
3. A witness search times out without finding $z_i = 1$. State what remains unknown, what additional evidence permits settlement, and whether the anchor construction is complete.

Part IV

Running the mechanism

Chapter 13

Partial outcomes and settlement

Restrict the payoff model when settlement information arrives, change the cost to quote known payoffs exactly, and prove that compatible settlements preserve the original uniform loss bound.

13.1 Partial outcomes and logical settlement

Suppose observation reveals some contract payoffs. A *partial outcome* σ is a set of fixed coordinate values consistent with at least one valid payoff vector. Its remaining payoff set and coherent set are

$$\mathcal{Z}_\sigma = \{z \in \mathcal{Z} : z_i = \beta_i \text{ for all } (i, \beta_i) \in \sigma\} \neq \emptyset, \quad \mathcal{M}_\sigma = \text{conv}(\mathcal{Z}_\sigma).$$

This is Definition 3.1 of the paper; its $\mathcal{I}_\sigma$ denotes the indices fixed by σ [P1, §3.2]. If new observations make the remaining set empty, the data, settlement rules, or model are inconsistent; projection to an empty set cannot resolve the problem.

Observation and logical implication are different sources of fixed values. For example, if $z_3 = z_1 z_2$ and observation fixes $z_1 = 0$, then logic fixes $z_3 = 0$ while z_2 may remain unresolved. Record which values were observed, which were deduced, and the evidence for each deduction.

To check whether $z_i = \beta$ is possible, solve a feasibility problem with that constraint or optimize the coordinate and use a valid bound. Finding a witness establishes possibility. Failure to find one before a timeout does not establish impossibility. Only a sound infeasibility or bound certificate permits logical settlement. The witness-averaging construction and source INTFW algorithm in Section 12.3 and Algorithm 3 use this rule [P1, §4.3].

13.2 Settlement changes the cost function

An unresolved LMSR market cannot quote exact endpoints at a finite state, as shown in Section 12.1. Once observation or a sound logical certificate fixes a payoff, change the cost by restricting its conjugate domain.

Let $V_\sigma = \{\mu : \mu_i = \beta_i \text{ for } (i, \beta_i) \in \sigma\}$ be the affine space fixing the settled coordinates. Define the convex indicator I_{V_σ} to be zero on V_σ and $+\infty$ outside, and set

$$R_\sigma = R + I_{V_\sigma}, \quad C_\sigma = R_\sigma^*.$$

Under the closed convex setup, this gives

$$C_\sigma(\theta) = \sup_{\mu \in V_\sigma} [\theta^\top \mu - R(\mu)]. \quad (13.1)$$

The indicator restricts candidate payoffs without changing R on the remaining domain; it is not a zero-one event indicator. This is the paper's Eq. (5), adapted from the information-preserving cost changes of Dudík, Frongillo, and Vaughan [P1, §3.2][P9].

For one categorical block, the restricted cost is

$$C_{j,\sigma}(\theta_j) = b \log \sum_{x \text{ still possible}} e^{\theta_{j,x}/b}.$$

If only category $x^\star$ remains, the cost is $\theta_{j,x^\star}$ and the price vector has a one at $x^\star$ and zeros elsewhere. This is the paper's partially settled LMSR example, with general liquidity retained [P1, Example 3.2].

For a numerical example, choose $b = 1$ and $\theta = (\log 0.2, \log 0.3, \log 0.5)$, so the original cost is zero and prices are $(0.2, 0.3, 0.5)$. If observation rules out category three,

$$C_\sigma(\theta) = \log(e^{\theta_1} + e^{\theta_2}) = \log 0.5, \quad p_\sigma(\theta) = (0.4, 0.6, 0).$$

The cost value drops by $\log 2$, but the switch itself transfers no cash.

In reduced binary form, settlement to zero gives cost $C_i(t) = 0$; settlement to one gives $C_i(t) = t$. Its fixed coordinate can leave the entropy-gradient calculation. Existing customer entitlements remain: a trader who bought one Yes share before settlement is still owed one dollar when Yes settles to one, or that dollar must be recorded as already paid. Removing a numerical variable cannot erase the liability. The holdings and cash records in Section 14.2 retain these payments.

13.2.1 What blockwise conditioning means

Within one categorical block, removing excluded categories and renormalizing has a conditioning interpretation. With overlapping blocks, this does not identify a unique prior joint distribution whose conditional marginals are all the new prices. Constraints and subsequent corrections enforce the required global consistency. Arbitrary probability clipping is neither a settlement event nor the prescribed conjugate-domain restriction.

13.3 A compatible sequence preserves the loss budget

The fixed-cost argument in Section 7.2 does not cover cost switches by itself. First consider real trades and compatible settlement switches, with no internal corrections. Initially $\sigma_0 = \emptyset$, state is θ^0 , and outside holdings and collected cash are zero. Fix one sequence $\sigma_0 \subseteq \dots \subseteq \sigma_T$ and one final payoff $z \in \mathcal{Z}$ compatible with it. Let θ^t be the state at switch t , and θ^{T+1} the final state.

At switch t , define the cost drop

$$J_t = C_{\sigma_{t-1}}(\theta^t) - C_{\sigma_t}(\theta^t) \geq 0. \quad (13.2)$$

The inequality follows because the new supremum in (13.1) is over a subset of the old domain. Real-trade charges telescope within each fixed-cost interval. Across the switches they give

$$K = C_{\sigma_T}(\theta^{T+1}) - C(\theta^0) + \sum_{t=1}^T J_t,$$

$$q = \theta^{T+1} - \theta^0.$$

Therefore actual loss is

$$L(z) = (\theta^{T+1} - \theta^0)^\top z - C_{\sigma_T}(\theta^{T+1}) + C(\theta^0) - \sum_{t=1}^T J_t.$$

Compatibility gives $R_{\sigma_T}(z) = R(z)$. Fenchel's inequality bounds $(\theta^{T+1})^\top z - C_{\sigma_T}(\theta^{T+1})$ by $R(z)$, and dropping the nonnegative cost-drop terms yields

$$L(z) \leq R(z) + C(\theta^0) - (\theta^0)^\top z \leq B(\theta^0).$$

This proves uniform validity of the original loss budget for every compatible sequence. It does not assert that the bound is tight on every observed path; information can exclude the originally worst outcome. The result corresponds to Proposition B.2 of the paper, whose original proof appears in the optional source concordance [P1, Appendix B].

An implementation that also makes internal corrections must include their accounting terms. At a correction Δ under the current cost, subtract its outcome-specific profit $\Delta^\top z - [C_\sigma(\theta + \Delta) - C_\sigma(\theta)]$ from the loss identity. A nonnegative worst-profit certificate makes this term nonnegative for every still-compatible final outcome. Thus these terms and the J_t terms both reduce the loss bound; the three records in Section 14.2 keep the real trades, virtual corrections, and cash distinct. The settlement-only proof above must not be applied to a changing-cost implementation while omitting those correction terms.

13.4 Optional source proof concordance

The source's statement below concerns preservation of the original worst-case budget across compatible information sequences. The main proof makes the uniform inequality explicit; it does not infer unchanged conditional tightness for each fixed information path. Source equation numbers (11)–(15) are retained for comparison. The state in the suprema of source Eq. (11) is written as θ_t , matching the state at the switch.

Proposition B.2 of the paper (proof in Appendix B). *If the initial market state is θ_0 then, regardless of the sequence of partial outcomes $\sigma_1, \dots, \sigma_T$, the worst-case loss of the market-maker using the sequence of costs C_{σ_t} is $\max_{\omega \in \Omega} D(\phi(\omega) \parallel \theta_0)$, i.e., the same as that of the market-maker using C without incorporating partial outcomes.*

The paper's θ_0 is θ^0 , and at $z = \phi(\omega)$ its $D(\phi(\omega) \parallel \theta_0)$ is $R(z) + C(\theta^0) - (\theta^0)^\top z = F_{\theta^0}(z)$. The common bound is therefore the loss budget $B(\theta^0)$ of Equation (7.1), which is Proposition B.1 of the paper (Chapter 7).

Proof, following Appendix B of the paper. The equation numbers (11)–(15) are the paper's. Recall that the market state at the time of switch from $C_{\sigma_{t-1}}$ to C_{σ_t} is θ_t . We first show that

the value of the cost at the time of switch decreases:

$$C_{\sigma_t}(\theta_t) = \sup_{\mu \in V_{\sigma_t}} [\theta_t \cdot \mu - R(\mu)] \leq \sup_{\mu \in V_{\sigma_{t-1}}} [\theta_t \cdot \mu - R(\mu)] = C_{\sigma_{t-1}}(\theta_t), \quad (11)$$

where the middle inequality follows because $V_{\sigma_t} \subseteq V_{\sigma_{t-1}}$. (**Independent check.** The manuscript prints $\theta \cdot \mu$ inside both suprema of Eq. (11); the state being evaluated is θ_t , as the outer expressions show, and that is what is written here.) We are now ready to prove the bound on the worst-case loss. Let $\Omega(\sigma_T)$ denote the set of outcomes compatible with σ_T , and recall that $\sigma_0 = \emptyset$, so $C_{\sigma_0} \equiv C$. Recall that θ_t for $t = 1, \dots, T$ are the states of the market when the cost becomes C_{σ_t} . Finally, let θ_{T+1} denote the final state. Then the worst-case loss of the market maker can be bounded as follows:

$$\begin{aligned} & \max_{\sigma_1 \subseteq \sigma_2 \subseteq \dots \subseteq \sigma_T} \max_{\omega \in \Omega(\sigma_T)} \sup_{\theta_1, \dots, \theta_{T+1}} \left[(\theta_{T+1} - \theta_0) \cdot \phi(\omega) - \sum_{t=0}^T (C_{\sigma_t}(\theta_{t+1}) - C_{\sigma_t}(\theta_t)) \right] \\ &= \max_{\sigma_1 \subseteq \sigma_2 \subseteq \dots \subseteq \sigma_T} \max_{\omega \in \Omega(\sigma_T)} \sup_{\theta_1, \dots, \theta_{T+1}} \left[(\theta_{T+1} - \theta_0) \cdot \phi(\omega) - (C_{\sigma_T}(\theta_{T+1}) - C_{\sigma_0}(\theta_0)) \right. \\ & \quad \left. - \sum_{t=1}^T (C_{\sigma_{t-1}}(\theta_t) - C_{\sigma_t}(\theta_t)) \right] \end{aligned} \quad (12)$$

$$\leq \max_{\sigma_T} \max_{\omega \in \Omega(\sigma_T)} \sup_{\theta_{T+1}} \left[(\theta_{T+1} - \theta_0) \cdot \phi(\omega) - C_{\sigma_T}(\theta_{T+1}) + C_{\sigma_0}(\theta_0) \right] \quad (13)$$

$$= \max_{\sigma_T} \max_{\omega \in \Omega(\sigma_T)} \left[R(\phi(\omega)) - \theta_0 \cdot \phi(\omega) + C(\theta_0) \right] \quad (14)$$

$$= \max_{\omega \in \Omega} D(\phi(\omega) \parallel \theta_0). \quad (15)$$

Eq. (12) follows by rearranging the terms. Eq. (13) follows by Eq. (11). Eq. (14) follows because the convex conjugate of C_{σ_T} is $R_{\sigma_T}(\mu) = \mathbb{I}\{\mu \in V_{\sigma_T}\} + R(\mu)$, and $R_{\sigma_T}(\phi(\omega)) = R(\phi(\omega))$ thanks to the compatibility of ω with σ_T . Finally, Eq. (15) follows from the definition of Bregman divergence, completing the proof. $\square$

Exercises.

1. Starting from prices (0.2, 0.3, 0.5), exclude category one instead of three. Compute the new cost, price vector, and cost drop. Does that drop enter the cash account?
2. A trader owns two Yes shares when Yes is settled to one. State the remaining payment obligation and the reduced binary cost after settlement.
3. For $z_3 = z_1 z_2$, compare observing $z_1 = 0$ with a timed-out search for $z_1 = 1$. Which coordinates can be fixed in each case, and why?
4. Add one certified internal correction to the displayed loss identity and verify the signs of its profit term and the settlement cost drop.

Chapter 14

The mechanism end to end and the three ledgers

Track outside holdings, collected cash, and the pricing state through real trades and internal corrections, and derive the resulting loss bound.

14.1 The Frank–Wolfe market maker, end to end

Paper. The mechanism charges each real trade using the current cost, incorporates newly observed or logically implied settlements, and attempts internal corrections. An inexpensive LCMM step uses selected logical constraints [P4]. PROJECTFW then searches for the Bregman projection onto the remaining coherent price set (Algorithm 2). Its accepted state has coherent prices and a nonnegative profit certificate; the finite stopping rule need not compute the exact minimizer. If no acceptable update has been established, it can retain the previous state, which may still admit arbitrage (Chapter 10). ■ §4, Mech. 1, p. 10

An accepted coherent update removes arbitrage at the current state. Exact optimality additionally identifies the maximum extractable guaranteed profit. These are different conclusions. On interruption, the certificate determines whether an update is available; an iteration count alone does not. The logical-settlement step also extends the partial outcome when it proves that other payoffs are fixed, allowing their prices to become exactly zero or one [P1, §4].

Source notation. The explicitly attributed mechanism above retains the paper’s symbols, including b as an integer-constraint vector. In the tutorial, liquidity is b , integer rows can be written $Az \geq a$, and relaxation rows $H\mu \geq h$. The mechanism’s ε_D is the small-objective tolerance ε_F ; its cost C_σ is the restricted cost in Equation (13.1). The stopping and interruption guarantees are those stated in Chapter 10, rather than an exact-projection guarantee.

The pricing state θ moves on every trade and on every internal step, but the last line pays each trader its own bundle: summed over trades the payout is $q^\top z$ with $q = \sum_t \delta_t$ and $z = \phi(\omega)$. Keeping these two quantities apart is the subject of the next section.

Mechanism 1 Frank–Wolfe Market Maker (FWMM), restated from Mechanism 1 of the paper

Input: cost function C , initial state θ_0 , initial partial outcome σ_0 ,
 LCMM constraints specified by $\tilde{A}, \tilde{b}$,
 IP constraints specified by A, b ,
 FW algorithm parameters $\alpha \in (0, 1), \varepsilon_0 \in (0, 1), \varepsilon_D > 0$
 Initialize the market state and partial outcome: $\theta \leftarrow \theta_0, \sigma \leftarrow \sigma_0$
for $t = 1, \dots, T$ (where T is an a priori unknown number of trades) **do**
 receive a request for a bundle δ_t
 sell the bundle δ_t for the cost $C_\sigma(\theta + \delta_t) - C_\sigma(\theta)$
 $\theta \leftarrow \theta + \delta_t$
 $\sigma \leftarrow \sigma \cup \{\text{newly settled securities if any}\}$
 perform an LCMM step:
 choose $\eta \geq 0$ such that $C_\sigma(\theta + \tilde{A}\eta) - C_\sigma(\theta) \leq \tilde{b} \cdot \eta$
 $\theta \leftarrow \theta + \tilde{A}\eta$
 perform a projection step:
 $(\sigma, \theta) \leftarrow \text{PROJECTFW}(\theta; C, \sigma, A, b, \alpha, \varepsilon_0, \varepsilon_D)$
end for
 Observe ω , consistent with σ
 Pay traders $\delta_1 \cdot \phi(\omega), \delta_2 \cdot \phi(\omega), \dots, \delta_T \cdot \phi(\omega)$

14.2 Separate three ledgers

The three principal records are outside holdings q , collected cash K , and pricing state θ . Assume initial holdings, cash, and internal offset are zero: $q^0 = 0, K^0 = 0$, and $v^0 = 0$. An internal correction Δ changes the pricing state without an outside trade. Its accumulated offset v is an auxiliary record, determined by

$$\theta = \theta^0 + q + v, \quad \text{equivalently} \quad q = \theta - \theta^0 - v, \quad (14.1)$$

where q denotes the aggregate shares held by outside traders (Section 7.2).

Implementation recommendation: Three principal records and an auxiliary offset.

The offset can be stored or derived from Equation (14.1) for an invariant check. The other three records determine prices and outside obligations.

Quantity	Meaning	Real trade δ	Internal correction Δ
q	Aggregate net shares held by outside traders	Add δ	No change
K	Cash collected from outside traders	Add $C(\theta + \delta) - C(\theta)$	No change
θ	State used in the pricing cost	Add δ	Add Δ
v	Accumulated internal correction offsets	No change	Add Δ

For a real trade, first save $\theta_{\text{old}} = \theta$ and compute $k = C(\theta_{\text{old}} + \delta) - C(\theta_{\text{old}})$. Then update $K \leftarrow K + k, q \leftarrow q + \delta$, and $\theta \leftarrow \theta_{\text{old}} + \delta$ as one transaction. Receipts can be negative

when the market pays a selling trader. For an internal correction Δ , update only θ and v . Record its hypothetical payoff-dependent profit for verification. At settlement, pay outside traders according to $q^\top z$, not $(\theta - \theta^0)^\top z$: settlement never uses the accumulated pricing-state change by itself.

14.2.1 Why safe internal trades preserve the bound

Derivation. For a fixed cost and a fixed outcome set, let the sequence of internal corrections be Δ_s , correction s being made from state θ_s . Define its hypothetical trader profit by

$$\pi_s(z) = \Delta_s^\top z - [C(\theta_s + \Delta_s) - C(\theta_s)].$$

Telescoping the cost changes gives

$$C(\theta_T) - C(\theta^0) = K + \sum_s [C(\theta_s + \Delta_s) - C(\theta_s)], \quad q = \theta_T - \theta^0 - \sum_s \Delta_s.$$

Substitute these expressions for cash and holdings into the actual loss:

$$\begin{aligned} L(z) &= q^\top z - K \\ &= (\theta_T - \theta^0)^\top z - C(\theta_T) + C(\theta^0) - \sum_s \pi_s(z). \end{aligned} \tag{14.2}$$

The first part has the original loss-budget bound: it is at most $B(\theta^0)$ (Equation (7.1)). Assume each correction has a verified lower bound $\pi_s(z) \geq \underline{\pi}_s \geq 0$ for every valid payoff vector. Then

$$L(z) \leq B(\theta^0) - \sum_s \underline{\pi}_s.$$

The accounting identity subtracts hypothetical correction profit from the base-mechanism loss. Actual cash and outside holdings remain unchanged during the correction; this subtraction does not represent a cash deposit.

With full outcome enumeration you can also compute the tighter bound

$$L(z) \leq B(\theta^0) - \min_{z' \in \mathcal{Z}} \sum_s \pi_s(z').$$

The minimum of the sum can exceed the sum of separate minima. For example, two corrections with outcome-profit vectors $(0, 1)$ and $(1, 0)$ each have minimum zero, while their cumulative profit is one in either outcome. The cumulative minimum evaluates all corrections at the same final outcome. The executable ledger test (`ledger_test`, Chapter 16) checks this accounting identity directly, including negative cash inflows for trades in which the market pays a selling trader.

For binary LMSR and an interior coherent candidate, take $\hat{\theta}_i = b \log[\mu_i / (1 - \mu_i)]$ and $\Delta = \hat{\theta} - \theta$. Define the exact Frank–Wolfe gap (Equation (9.2))

$$g = \Delta^\top \mu - \min_{z \in \mathcal{Z}} \Delta^\top z.$$

This gap upper-bounds objective suboptimality. Conjugate equality then gives $\min_z \pi(z) = F_\theta(\mu) - g$, the certificate of Equation (10.1). A safe correction requires this value, or a valid lower bound on it, to be nonnegative. Coherence alone does not establish that requirement [P1, §4.2 and App. D].

This derivation holds a fixed cost. Section 13.3 treats the cost changes that settlement makes.

14.2.2 A real trade followed by an internal correction

Consider two identical binary contracts with $b = 1$ and initial state $\theta^0 = (0, 0)$. A trader buys the signed bundle $\delta = (\log 4, \log(2/3))$. The market collects

$$K = C(\delta) - C(0) = \log(25/12) \approx 0.733969.$$

Prices become $(0.8, 0.4)$. The optimal correction moves the pricing state to

$$\theta' = (\tfrac{1}{2} \log(8/3), \tfrac{1}{2} \log(8/3)) \approx (0.490415, 0.490415).$$

Record	After the real trade	After the correction
Outside holdings q	$(1.386294, -0.405465)$	Unchanged
Collected cash K	0.733969	Unchanged
Pricing state θ	$(1.386294, -0.405465)$	$(0.490415, 0.490415)$
Offset v	$(0, 0)$	$(-0.895880, 0.895880)$

The invariant [Equation \(14.1\)](#) holds before and after the correction. Actual loss is approximately -0.733969 if the common event fails and 0.246860 if it occurs. These losses do not change at the instant of the correction because neither cash nor outside holdings changes. The correction changes prices for future trades and the hypothetical-profit term in the bound-preservation identity.

14.3 A coherent state is not an always-coherent transaction rule

One could define a different mechanism that charges differences of $C_{\mathcal{M}}$ on every trade, where $C_{\mathcal{M}}(\theta) = \sup_{\mu \in \mathcal{M}} \{\theta^\top \mu - R(\mu)\}$ is the constrained conjugate cost of [Section 6.4.1](#). Its optimizing payoff vector is coherent. Where this optimizer is unique and the cost is differentiable, it is the price gradient; otherwise prices require an explicit optimizer or subgradient selection. Computing them requires solving the constrained optimization problem at each required state.

The paper's mechanism instead uses the inexpensive base cost C for real trades and periodically changes the internal state ([Mechanism 1](#)). Even if prices are coherent immediately after a correction, the next independent-market trade can move them outside $\mathcal{M}$. Correction scheduling is therefore part of the mechanism's behavior and of its experimental evaluation ([Chapter 17](#)).

Exercises.

1. In [Section 14.2.2](#), verify both outcome losses and the invariant after each operation. Then add a real trade $(1, 0)$, compute its charge from the pre-trade state, and verify [Equation \(14.2\)](#).
2. For correction-profit vectors $(0, 1)$ and $(1, 0)$, compare the sum of separate minima with the minimum cumulative profit. Identify the outcome used in each calculation.

Part V

A worked example

Chapter 15

The triangle market: an analytic projection

Understand the six-security example, derive its exact coherent projection, and interpret the historical validation results.

Use the payoff order $(x_1, x_2, x_3, x_1x_2, x_2x_3, x_1x_3)$ from [Chapter 8](#), with independent binary LMSR costs and common liquidity $b > 0$. Write $\text{logit}(r) = \log(r/(1-r))$ for $0 < r < 1$. The starting prices and state are

$$p^0 = (1/2, 1/2, 1/2, 1/20, 1/20, 1/20), \quad \theta_i^0 = b \text{logit}(p_i^0).$$

For this starting point, the triangle has an exact analytic projection: $a = (1 + \sqrt{7})/9$ for each singleton and $r = a - 1/3$ for each conjunction. We construct this candidate, exhibit a probability distribution with those means, and prove global optimality. The result supplies an independent target for the numerical experiment in [Chapter 16](#).

15.1 The analytic projection

Derivation. Construct a symmetric candidate

$$\mu^\star = (a, a, a, r, r, r).$$

on the face $3a - 3r = 1$, so that $r = a - 1/3$. This is a choice of candidate, not a deduction that every violated constraint becomes active at a projection. We will verify that the resulting point solves the full problem. Along this face, for $1/3 < a < 1$, define

$$f(a) = 3b [\text{KL}(\text{Bern}(a) \parallel \text{Bern}(1/2)) + \text{KL}(\text{Bern}(a - 1/3) \parallel \text{Bern}(1/20))].$$

The derivative of Bernoulli KL with respect to its first argument is the difference of logits. Since $d(a - 1/3)/da = 1$, the chain rule gives

$$\frac{f'(a)}{3b} = \text{logit}(a) + \text{logit}(a - 1/3) - \log(1/19).$$

Setting this expression to zero yields

$$19a(a - 1/3) = (1 - a)(4/3 - a), \quad 27a^2 - 6a - 2 = 0.$$

The root $(1 - \sqrt{7})/9$ is negative and outside the domain. The admissible root is

$$a = \frac{1 + \sqrt{7}}{9} \approx 0.4050834790, \quad r = a - \frac{1}{3} \approx 0.0717501457. \quad (15.1)$$

To establish feasibility, assign probability $s = 2/3 - a$ to each of the three outcomes with exactly one underlying bit equal to one, and probability r to each outcome with exactly two bits equal to one. Give zero mass to 000 and 111. Both s and r are positive and $3(s + r) = 1$. Each singleton is present in one one-bit outcome and two two-bit outcomes, so its mean is $s + 2r = a$. Each conjunction is present in exactly its corresponding two-bit outcome, so its mean is r . Thus all six means equal $\mu^\star$, proving feasibility.

To establish global optimality, write $n = (1, 1, 1, -1, -1, -1)$. The candidate's singleton gradient entries are $b \logit(a)$; its conjunction entries are $b[\logit(r) - \log(1/19)] = -b \logit(a)$ by the stationary equation. Consequently

$$\nabla F_{\theta^0}(\mu^\star) = -\kappa n, \quad \kappa = -b \logit(a) > 0.$$

Every coherent μ has $n^\top \mu \leq 1 = n^\top \mu^\star$. Therefore

$$\nabla F_{\theta^0}(\mu^\star)^\top (\mu - \mu^\star) = \kappa(1 - n^\top \mu) \geq 0,$$

which proves global optimality by convexity. This is stronger than solving only the symmetric one-dimensional slice and assuming its answer is globally optimal.

The optimal guaranteed arbitrage value at $b = 1$ is

$$F_{\theta^0}(\mu^\star) \approx 0.06762791589185.$$

The optimum lies on a face of $\mathcal{M}$ while every marginal remains strictly between zero and one. It therefore has a finite inverse LMSR state. A boundary point of the coherent price set is not necessarily a singularity of the entropy gradient (compare [Chapter 12](#)).

15.2 Observed default results

Default parameters are $b = 1$ and $\alpha = 0.9999$. These are observations from the original guide’s historical run, not results copied from the paper or a claim of a new run. The unchanged program is printed in [Appendix C](#); commands and output meanings are in [Chapter 16](#). The original raw JSON was not preserved. The following values retain the recorded precision, which is not a formal numerical error bound.

Table 15.1: Observed results of the default run ($b = 1$, $\alpha = 0.9999$), recorded in the original guide.

Check	Result
<i>Analytic target and initial-state checks</i>	
Local constraints at the initial prices	All satisfied
Initial numerical hull-membership flag	False
Triangle-inequality violation	0.35
LP oracle value for $-\sum x + \sum y$	-1.5
Integer oracle value for the same objective	-1.0
Analytic optimal arbitrage value	0.06762791589
<i>Observed numerical iterate</i>	
Frank–Wolfe iterations in this run	14
Final Frank–Wolfe singleton prices	Approximately 0.40508144 each
Final Frank–Wolfe conjunction prices	Approximately 0.07175071 each
Final $F_{\theta^0}(\mu)$ against the original state	0.06763091685
<i>Certificate and validation checks</i>	
True gap checked by enumeration	0.00000302942
Worst actual outcome profit of the correction	0.06762788743
Guarded numerical lower certificate	0.06762711998
Worst profit of arbitrary coherent reset	-0.70916633419
Finite-difference gradient maximum error	Approximately 1.53×10^{-10}

At the displayed precision, the objective minus the enumerated gap is

$$0.06763091685 - 0.00000302942 = 0.06762788743.$$

This is the worst outcome profit $F_{\theta^0}(\mu) - g$, rather than the objective alone. The guarded lower certificate $F_{\theta^0}(\mu) - \bar{g} = 0.06762711998$ is smaller because the guarded oracle lower bound makes $\bar{g}$ at least as large as the true gap when the bound is valid.

For the arbitrary reset, the target is $(0.5, 0.5, 0.5, 0.25, 0.25, 0.25)$, the marginals of independent fair underlying bits. It is coherent, but the move from the initial market gives a negative worst-case internal-trade profit.

Interrupted projection fixture. These runs use the same nonuniform starting prices and $\alpha = 0.9999$. Their iteration counts are historical regression observations:

- One iteration returns the original state because no safe candidate has been established.
- Seven iterations return a coherent safe state with minimum profit about 0.06222049, without claiming the final requested extraction accuracy.

These iteration counts are specific to the recorded runs. Solver choices and initialization can change when the first safe candidate appears.

Ledger fixture. The ledger experiment starts separately from $\theta^0 = 0$ and uses $\alpha = 0.95$. It checks all eight outcomes after three real trades. The original bound (Equation (7.1)) is $6 \log 2 \approx 4.15888308$; the final worst actual loss is approximately 2.14800263, below both that bound and the verified tightened bound of approximately 4.08727940.

15.3 Numerical and algorithmic scope

The enumerated reference uses a tiny 10^{-12} contraction and a numerical convex optimizer. The analytic formula supplies the independent exact reference. The MIP oracle is solved to its reported optimal status, with an explicit floating-point allowance subtracted from its bound; all 66 oracle calls in the default program are then audited by enumeration. This count includes initialization, the main projection, interruption runs, and ledger checks; it is not the number of oracle calls in the 14-iteration main projection alone.

Caution. The code uses this guarded upper gap (Equation (11.1)) in its contraction rule. That is a conservative implementation modification, not a verbatim implementation of Algorithm 2 of the paper (Algorithm 2) or a proof that its original convergence theorem applies unchanged. The script is intended for moderate parameters in this tiny model. Very small divergences, extreme logits, or arbitrarily strict α values require further numerical treatment. The program does not implement the tournament language, full general initialization, an LCMM prepass, settlement updates, historical replay, or concurrent trading. Its results validate this small mechanism example and do not establish scalability.

Exercises.

1. Reconstruct the six nonzero outcome weights from a and r . Check their signs, sum, singleton means, and conjunction means.
2. Using Table 15.1, compute the difference between the analytic optimal value and the observed worst profit. Explain why the numerical objective itself can exceed the analytic minimum.

Chapter 16

Running and auditing the code

Run the triangle experiment and audit the inputs, failure behavior, and numerical evidence behind each reported certificate.

This chapter distinguishes two supplied programs and one proposed system:

- The [historical Python module](#) is the unchanged source of every listing below and of the recorded results in [Section 15.2](#).
- The separate [hardened triangle module](#) adds numerical validation and clearer return diagnostics. Its capabilities, limits, and commands are given in [Section 16.13](#).
- The general engine in [Chapter 19](#) remains a proposal. Neither triangle program implements its settlement, deadline, or atomic commit interfaces.

The walkthrough first audits the historical program. To run it from the tutorial directory:

```
| python code/fwmm_triangle.py > results.json
```

[Appendix C](#) prints that file in full. Its content hash and the current verification environment are recorded separately in [the revision manifest](#); new check results belong to [the independent check results](#). Neither record changes the provenance of the historical values in [Section 15.2](#).

16.1 What a successful run demonstrates

The program audits the IP payoff model against all 64 binary six-vectors, compares LP and IP minima, computes an enumerated and analytic projection, checks its gradient, verifies correction profits over all eight outcomes, and runs interruption and cash/liability checks. It emits a JSON object after the assertions pass. At $b = 1$ the independent analytic target is $(a, a, a, a - 1/3, a - 1/3, a - 1/3)$ with $a = (1 + \sqrt{7})/9$ ([Equation \(15.1\)](#)), and optimal arbitrage value approximately 0.06762791589185.

Caution: scope. This is a tiny model with an enumeration audit, a pragmatic floating-point allowance on MIP bounds, and an engineering modification to the contraction rule. It is not a scalability result or a complete tournament engine. It does not implement full general initialization, LCMM, settlement, replay, or concurrent trading. Extreme logits, very small divergences, or arbitrarily strict extraction targets need further numerical treatment. The derivation and the historical output table are in

Chapter 15; the executable and its assumptions are complete here and in [Appendix C](#).

16.2 Environment and commands

Status. The historical script was run with **Python 3.12.14, NumPy 2.3.5, and SciPy 1.17.0**. Use Python 3.12 and these package versions for the closest reproduction. The experiment uses SciPy’s HiGHS-backed milp, so it does not require Gurobi [S1].

From the tutorial directory, create an environment:

```
| python -m venv .venv
```

Activate it on macOS/Linux:

```
| source .venv/bin/activate
```

Or on Windows PowerShell:

```
| .\.venv\Scripts\Activate.ps1
```

Install and run:

```
| python -m pip install numpy==2.3.5 scipy==1.17.0
| python code/fwmm_triangle.py > results.json
| python code/fwmm_triangle.py --b 10 > results_b10.json
| python code/fwmm_triangle.py --alpha 0.5 > results_alpha05.json
```

Caution. The script prints JSON and runs its checks through ordinary Python assertions, so do not run it with `python -O`.

These package versions describe the original run, not an assurance that the current interpreter or native solver matches it. A reproduction record must include the script hash, command, platform, Python and package versions, and the available HiGHS/native-solver version; if a native version cannot be identified, record that limitation. The original raw JSON was not preserved.

Status. Additional successful runs reported during the original preparation covered $b = 0.1$, $b = 10$, and $\alpha = 0.5$. A fresh environment or solver build can alter the final digits or iteration count; the invariants and analytic target are the relevant comparison.

16.3 Function contracts and evidence

Each function is described by its inputs, mathematical output, conditions for that output, failure behavior, and associated check. Line numbers refer to the unchanged artifact and [Appendix C](#). Some comments in that historical file use stronger language than its numerical contract warrants; the text below identifies those limits rather than silently changing the listing.

16.4 Header and integer payoff model

The file opens with its docstring and imports (lines 1–15), followed by the module constants that define the integer payoff model (lines 18–34).

```

1  #!/usr/bin/env python3
2  """Pedagogical FWMM experiment; NumPy/SciPy only, not an NCAA replication.
3
4  Usage: python fwmm_triangle.py [--b 1.0] [--alpha 0.9999]
5  Prints JSON containing numerical checks, the FW trace, and the final
6  ↪ certificate.
7  """
8  import argparse
9  import itertools
10 import json
11 import platform
12
13 import numpy as np
14 import scipy
15 from scipy.optimize import Bounds, LinearConstraint, linprog, milp, minimize
16 from scipy.special import expit, xlogy

```

Listing 16.1: Docstring and imports

NumPy and SciPy are the direct Python dependencies. Their native libraries and platform also matter to reproduction. SciPy’s `milp` supplies the integer oracle, `linprog` the numerical LP checks, and `minimize` the restricted-master solve. The elementary functions `expit` and `xlogy` provide the logistic function and the convention $0 \log 0 = 0$.

```

18 PAIRS = [(0, 1), (1, 2), (0, 2)]
19 N = 6 # x1, x2, x3, x1*x2, x2*x3, x1*x3; each is a Yes security
20 A, rhs = [], []
21 for k, (i, j) in enumerate(PAIRS, start=3):
22     for parent in (i, j):
23         row = np.zeros(N)
24         row[k], row[parent] = 1, -1
25         A.append(row)
26         rhs.append(0.0)
27     row = np.zeros(N)
28     row[i], row[j], row[k] = 1, 1, -1
29     A.append(row)
30     rhs.append(1.0)
31 A, rhs = np.array(A), np.array(rhs)
32 BOUNDS = Bounds(np.zeros(N), np.ones(N))
33 CONSTRAINTS = LinearConstraint(A, -np.inf, rhs)
34 ORACLE_AUDIT = []

```

Listing 16.2: Module constants: security order and the integer payoff model

The six coordinates are the Yes securities $x_1, x_2, x_3, y_{12}, y_{23}, y_{13}$ (line 19). For each pair (i, j) in `PAIRS`, with its product stored at index $k = 3, 4, 5$, the loop writes three rows of $Az \leq \text{rhs}$:

$$y_{ij} - x_i \leq 0, \quad y_{ij} - x_j \leq 0, \quad x_i + x_j - y_{ij} \leq 1.$$

These are the AND inequalities of the triangle example (Chapter 8). Together with the bounds $0 \leq z \leq 1$ in `BOUNDS` and integrality they give an integer description of $\mathcal{Z}$ in the sense of Equation (8.1), written with $\leq$ rows. The list `ORACLE_AUDIT` collects every oracle objective together with the lower bound returned for it, so that `main` can audit all bounds by enumeration at the end (lines 253–254).

Independent check. The first assertion of `main` (line 197) confirms that exactly the eight valid payoff vectors, and no other binary six-vector, satisfy these rows: this is the audit

of the IP payoff model against all 64 binary six-vectors. It checks both inclusions: every enumerated payoff satisfies the rows, and every admitted binary assignment is one of those payoffs. The coordinate order is part of this contract; a conjunction-coordinate mismatch changes the set being tested.

16.5 The integer oracle

Every integer solve in the program goes through one function.

```

37 def oracle(c):
38     """Exact-to-solver-tolerances integer LMO; fail closed without
    ↪ optimality."""
39     scale = max(float(np.max(abs(c))), 1e-100)
40     result = milp(c/scale, integrality=np.ones(N), bounds=BOUNDS,
41                  constraints=CONSTRAINTS,
42                  options={"mip_rel_gap": 0.0, "presolve": True})
43     if result.status != 0 or result.x is None:
44         raise RuntimeError("Oracle did not prove optimality: " +
    ↪ result.message)
45     z = np rint(result.x)
46     if (np.max(abs(z - result.x)) > 1e-7 or
47         np.max(A @ z - rhs) > 1e-7):
48         raise RuntimeError("Invalid integer oracle solution")
49     # Floating-point MIP optimality has tolerances. This guard is a pragmatic
50     # numerical allowance, not an exact-arithmetic theorem. Every returned
51     # bound is subsequently audited by enumeration in this tiny experiment.
52     lower_bound = scale * (float(result.mip_dual_bound)-2e-6)
53     incumbent = float(c @ z)
54     ORACLE_AUDIT.append((c.copy(), lower_bound))
55     return z, lower_bound, incumbent

```

Listing 16.3: Historical integer oracle requiring solver-reported optimality

`oracle(c)` solves the linear minimization $\min_{z \in \mathcal{Z}} c^\top z$ of Equation (9.1) with SciPy’s HiGHS-backed `milp` [S1]. The objective is divided by $s = \max\{\|c\|_\infty, 10^{-100}\}$ before the solve, with zero objective offset, and the bound is multiplied by s afterwards; `mip_rel_gap` is set to zero. The floor also covers $c = 0$. Inputs are a finite coefficient vector of length six and the fixed binary model above. The routine requires solver-reported optimal status (status zero) and a returned point; otherwise it raises. It checks integrality tolerance and logical rows after rounding. The historical docstring’s “Exact-to-solver-tolerances” is not an exact-arithmetic guarantee.

The returned lower bound is the solver’s dual bound minus an explicit allowance of 2×10^{-6} in scaled units, or $2 \times 10^{-6}s$ in the original objective units. In the code’s words, floating-point MIP optimality has tolerances, and this guard is a pragmatic numerical allowance, not an exact-arithmetic theorem. The function returns the vertex z , the guarded lower bound m_{lb} , and the incumbent value $c^\top z$. The lower bound gives an upper gap $\bar{g}$ by Equation (11.1) only if it is indeed no greater than the true minimum. Solver status and the fixed allowance do not prove that inequality for arbitrary models.

Status. Every returned bound is subsequently audited by enumeration in this tiny experiment. In the recorded default run all 66 oracle calls were audited in this way (Section 15.3).

Failure contract and scope. This function does not implement the time-limited-bound interface of [Chapter 11](#): nonoptimal statuses are rejected. The outer loop’s iteration cap is checked between completed solves and cannot interrupt an in-progress MIP call. A wall-clock deadline needs an in-solver time limit and a policy for validating any returned bound.

The historical routine omits some explicit checks for nonfinite inputs, missing bounds, and rounded solutions outside the variable bounds. A general oracle interface must validate dimensions, finite coefficients and bounds, integrality, variable bounds, and logical rows before certification. Its rejection tests should include NaN, infinity, missing bounds, and invalid rounded points. These requirements do not imply that the historical default run encountered those failures. [Section 16.13](#) describes the separate hardened triangle module and the limits of its checks.

16.6 Cost, convex conjugate, state, and correction objective

Four short functions implement the reduced binary LMSR of the local setup and its convex conjugate.

```

58 def cost(theta, b):
59     return float(b * np.logaddexp(0.0, theta / b).sum())
60
61
62 def entropy(mu, b):
63     # xlogy defines 0*log(0)=0, so entropy is continuous at endpoints.
64     return float(b * (xlogy(mu, mu) + xlogy(1-mu, 1-mu)).sum())
65
66
67 def state(mu, b):
68     # Never clip: endpoint prices require a different, settled cost function.
69     if np.any(mu <= 0) or np.any(mu >= 1):
70         raise ValueError("Finite state requires every marginal strictly in
71             ↪ (0,1)")
72     return b * (np.log(mu) - np.log1p(-mu))
73
74 def divergence(mu, theta, b):
75     return entropy(mu, b) + cost(theta, b) - float(theta @ mu)

```

Listing 16.4: Reduced binary LMSR cost, its convex conjugate, the inverse state map, and the correction objective

- **cost** is $C(\theta) = b \sum_i \log(1 + e^{\theta_i/b})$ ([Equation \(2.4\)](#)), evaluated with `logaddexp`.
- **entropy** is the convex conjugate (scaled negative entropy) $R(\mu) = b \sum_i [\mu_i \log \mu_i + (1 - \mu_i) \log(1 - \mu_i)]$ ([Equation \(4.4\)](#)). As the comment says, `xlogy` defines $0 \log 0 = 0$, so the function is continuous at the endpoints.
- **state** is the inverse map $\theta = \nabla R(\mu) = b \operatorname{logit}(\mu)$. It never clips: a marginal at 0 or 1 raises an error, because endpoint prices require a different, settled cost function ([Chapter 13](#)).
- **divergence** is the correction objective $F_\theta(\mu) = R(\mu) + C(\theta) - \theta^\top \mu$ ([Equation \(4.3\)](#)). Its gradient in μ is $\operatorname{state}(\mu) - \theta = \nabla R(\mu) - \theta$, the oracle direction c .

Function	Mathematical inputs	Output and limits
cost	finite $\theta \in \mathbb{R}^6$, finite $b > 0$	scalar cost
entropy	$\mu \in [0, 1]^6$, finite $b > 0$	negative entropy $R(\mu)$
state	$\mu \in (0, 1)^6$, finite $b > 0$	finite inverse state
divergence	compatible vectors, $\mu \in [0, 1]^6$	$F_\theta(\mu)$; its gradient requires interior μ

These are domain requirements, not a claim that every one is checked by the historical functions. A general API must validate shapes, finite values, positive liquidity, and domain membership explicitly. In particular, endpoint rejection does not detect every invalid input.

The expression for divergence can subtract nearly equal quantities near the current prices. Stable elementary functions help but do not remove that cancellation. A refinement should test nearly equal prices against an independent evaluation and use an explicit error budget. Silent clipping of a materially negative objective would hide inconsistent data.

Independent check. Central differences in `main` (lines 217–223) check this derivative, the one used by the linear minimization oracle, at a test point.

16.7 The fully corrective step

Fully corrective Frank–Wolfe optimizes convex-mixture weights on feasible outcome columns; this function does that optimization.

```

78 def correct(active, u, eps, theta, b, initial_weights=None):
79     """Minimize over the convex hull of contracted active vertices."""
80     contracted = (1-eps) * np.asarray(active) + eps * u
81     k = len(contracted)
82     w0 = np.full(k, 1/k) if initial_weights is None else initial_weights
83     objective = lambda w: divergence(w @ contracted, theta, b)
84     gradient = lambda w: contracted @ (state(w @ contracted, b) - theta)
85     result = minimize(objective, w0, jac=gradient, method="SLSQP",
86                      bounds=Bounds(np.zeros(k), np.ones(k)),
87                      constraints=LinearConstraint(np.ones((1, k)), 1, 1),
88                      options={"ftol": 1e-13, "maxiter": 2000})
89     if not result.success:
90         raise RuntimeError("Active-set solve failed: " + result.message)
91     # Repair only solver roundoff, retaining an explicitly feasible mixture.
92     if min(result.x) < -1e-8 or abs(sum(result.x)-1) > 1e-8:
93         raise RuntimeError("Invalid active-set weights")
94     w = np.maximum(result.x, 0)
95     w /= w.sum()
96     return w @ contracted, w

```

Listing 16.5: Fully corrective step over the contracted active vertices

`correct` minimizes F_θ over the convex hull of the contracted active vertices $(1-\varepsilon)z + \varepsilon u$, $z \in \mathcal{Z}_t$ (Equation (12.1)). It optimizes the convex-mixture weights w on these columns, not μ directly: the rows of $V_\varepsilon \in \mathbb{R}^{k \times 6}$ are the contracted vertices, $w \in \mathbb{R}^k$, and $\mu = V_\varepsilon^\top w$. The chain rule gives

$$dF_\theta(\mu) = (\nabla R(\mu) - \theta)^\top V_\varepsilon^\top dw, \quad \nabla_w F_\theta(V_\varepsilon^\top w) = V_\varepsilon (\nabla R(\mu) - \theta).$$

SLSQP solves the problem under $0 \leq w \leq 1$ and $\sum w = 1$; a failed solve raises. Afterwards the code repairs only solver roundoff, retaining an explicitly feasible mixture: weights more

negative than -10^{-8} , or a sum that misses one by more than 10^{-8} , raise an error; otherwise negative entries are set to zero and the weights are renormalized.

The output is the repaired mixture and its weights. A nonnegative normalized weight vector still represents a coherent mixture of coherent atoms; clipping price coordinates independently would not preserve that representation. Numerical solver success does not prove an exact restricted-master optimum. The outer loop therefore recomputes the objective, gradient, and global certificate at the actual repaired mixture. A general master interface must also reject nonfinite weights and malformed warm starts before normalization; its tests should check the weight gradient by finite differences and reject transposed active arrays. The historical gradient test in `main` checks the objective's price gradient, not all of these API conditions. The separate hardened module adds mixture validation and weight-gradient tests (Section 16.13).

The same function, called by `main` with all eight vertices and contraction 10^{-12} (line 210), produces the enumerated reference projection.

16.8 The Frank–Wolfe loop

`project_fw` is adaptive contracted fully corrective Frank–Wolfe using integer oracle calls; compare Algorithm 2 of the paper as restated in Algorithm 2. It is listed in two parts.

```

99 def project_fw(theta, b, alpha, max_iterations=200):
100     # For this specific model, these two valid vertices cover 0 and 1 in
101     # every coordinate; their mean suffices for the paper's coordinate
102     # interior condition. A general model needs the full InitFW routine.
103     active = [oracle(np.ones(N))[0], oracle(-np.ones(N))[0]]
104     u = np.mean(active, axis=0)
105     eps, weights, best, trace = 0.1, None, None, []
106     stop_reason = "iteration budget"
107     for iteration in range(1, max_iterations+1):
108         mu, weights = correct(active, u, eps, theta, b, weights)
109         new_theta = state(mu, b)
110         c = new_theta - theta
111         z, lower_bound, incumbent = oracle(c)
112         f = divergence(mu, theta, b)
113         # A minimization lower bound gives an UPPER bound on the FW gap.
114         g_upper = max(0.0, float(c @ mu) - lower_bound)
115         g_incumbent = float(c @ mu) - incumbent
116         profit_bound = f - g_upper
117         record = dict(iteration=iteration, active_vertices=len(active),
118                       contraction=eps, divergence=f, gap_upper=g_upper,
119                       gap_incumbent=g_incumbent,
120     ↪ profit_lower_bound=profit_bound)
121         trace.append(record)
122         if best is None or profit_bound > best["profit_lower_bound"]:
123             best = dict(record, prices=mu.copy(), theta=new_theta.copy(),
124                           weights=weights.copy(),
125     ↪ active=np.asarray(active).copy())

```

Listing 16.6: `project_fw`, part 1: initialization, correction, oracle call, and certificate

Initialization (lines 100–106). The oracle with objectives ± 1 returns the all-zeros and the all-ones payoff vectors. In the code's words, for this specific model these two valid vertices cover 0 and 1 in every coordinate, and their mean suffices for the paper's coordinate

interior condition; a general model needs the full INITFW routine (Algorithm 3). The mean is $u = (1/2, \dots, 1/2)$ and the contraction starts at $\varepsilon = 0.1$. Coordinate interior does not imply relative interior of the whole coherent polytope: this anchor is supported on only those two outcomes.

Each iteration (lines 107–123) solves the fully corrective subproblem for μ , maps it to the candidate state $\hat{\theta} = \text{state}(\mu)$, and sets the oracle direction $c = \hat{\theta} - \theta = \nabla R(\mu) - \theta$. One oracle call returns a vertex z , the guarded lower bound m_{lb} , and the incumbent value. A minimization lower bound gives an *upper* bound on the Frank–Wolfe gap, so $c^\top \mu - m_{\text{lb}}$ is the upper gap $\bar{g}$ of Equation (11.1) when the bound is valid. The historical code clamps this expression to zero before computing `profit_bound` = $F_\theta(\mu) - \bar{g}$. A materially negative raw upper gap indicates an invalid bound, inconsistent candidate, or numerical failure; clamping conceals that diagnostic. A general interface must reject material negativity, document its allowance for tiny roundoff, and validate the bound before a live-state commit. The later enumeration audit limits this issue in the offline example but does not justify the pattern for an online engine. The separate hardened triangle module checks the bound before using a certificate (Section 16.13). The gap computed from the incumbent, `g_incumbent`, is written to the trace but enters no decision. Each iteration appends a record to the trace, and the iteration with the best certificate so far is kept together with its prices, state, weights, active set, and contraction used to interpret those weights. A returned best candidate can precede the final iteration; all snapshot fields must describe that same candidate.

```

124         if g_upper <= (1-alpha)*f or f <= 1e-12:
125             stop_reason = "accuracy threshold"
126             break
127         if not any(np.array_equal(z, old) for old in active):
128             active.append(z)
129             weights = np.append(weights, 0.0)
130         gu = float(c @ (mu-u))
131         # Using the guarded upper gap here is an engineering modification,
132         # not a verbatim Algorithm 2 implementation or a convergence proof.
133         if gu < 0 and g_upper / (-4*gu) < eps:
134             eps = min(g_upper / (-4*gu), eps/2)
135         if best["profit_lower_bound"] < 0:
136             return theta.copy(), best, trace, "unchanged: " + stop_reason
137         return best["theta"], best, trace, "safe update: " + stop_reason

```

Listing 16.7: `project_fw`, part 2: stopping rule, column addition, contraction update, and the safe return

The historical label `accuracy threshold` merges two conditions: $\bar{g} \leq (1 - \alpha)F_\theta(\mu)$ (Equation (10.2)) establishes an extraction target, whereas $F_\theta(\mu) \leq 10^{-12}$ establishes a small upper bound on remaining arbitrage value. The latter permits retaining the original state and does not establish the extraction fraction. Otherwise the oracle vertex joins the active set if it is new, with weight zero. The contraction is then updated: with $g_u = c^\top(\mu - u)$, if $g_u < 0$ and $\bar{g}/(-4g_u) < \varepsilon$, the contraction becomes $\min\{\bar{g}/(-4g_u), \varepsilon/2\}$.

Caution. The code uses this guarded upper gap in its contraction rule. That is a conservative implementation modification, not a verbatim implementation of Algorithm 2 of the paper or a proof that its original convergence theorem applies unchanged.

The return (lines 135–137) tests the computed certificate, whose guarantee depends on the numerical conditions above. If the best certificate is negative, the original state is returned with an action beginning `unchanged`; otherwise the best state is returned as

a safe update. Either way the action names the stop reason, accuracy threshold or iteration budget.

A general return interface should record the trigger separately from the action:

- `extraction_target`: the fraction-of-profit inequality holds;
- `small_arbitrage_value`: the objective is below its tolerance;
- `iteration_budget`: the outer iteration cap is reached.

A genuine solver deadline is another trigger, absent here. A zero-iteration budget also needs an explicit no-candidate return; the preserved routine otherwise accesses `best` before it exists. The hardened triangle module already separates actions from stopping reasons and handles a zero-iteration return; it does not implement a solver deadline.

Independent check. The interruption tests in `main` exercise this rule in the recorded fixture. One iteration returns the original state because no safe candidate has been established. Seven iterations return a coherent safe state without claiming the final requested extraction accuracy (Section 15.2). These counts are version-specific observations, not mechanism-independent guarantees about when a safe candidate appears.

16.9 Numerical hull membership over enumerated outcomes

The convex-mixture formulation exactly characterizes membership in $\mathcal{M}$; the solver evaluates it numerically.

```

140 def membership(mu, vertices):
141     result = linprog(np.zeros(len(vertices)),
142                     A_eq=np.vstack([np.ones(len(vertices)), vertices.T]),
143                     b_eq=np.r_[1, mu], bounds=(0, None), method="highs")
144     return bool(result.success)

```

Listing 16.8: Historical Boolean result from a numerical feasibility LP

`membership` takes a six-vector and an array of enumerated payoff vertices. It asks `linprog` whether nonnegative weights summing to one reproduce μ as a mixture of the explicitly enumerated vertices; the objective is zero, so the LP is a pure feasibility test. `main` uses it four times: to show that p^0 , which satisfies every local row, is not coherent (line 202), and to confirm that the final Frank–Wolfe prices (line 231), the arbitrary reset target (line 245), and the interrupted seven-iteration state (line 252) are coherent.

Its Boolean `result.success` conflates reported infeasibility with limits and numerical failures. A general interface needs three outcomes: feasible with verified nonnegative weights and equality residuals; infeasible according to solver status; and indeterminate. Numerical failure must not imply mathematical nonmembership. For this initial point, the independent analytic separator gives $\sum_i p_i^0 - \sum_{i < j} p_{ij}^0 = 1.35 > 1$, which proves nonmembership without an LP status.

16.10 The ledger test

The ledger test runs three real trades through the market and corrects the internal state after each one.

```

147 def ledger_test(vertices, b, alpha):
148     """Actual trader liabilities q and cash are separate from internal
    ↪ theta."""
149     theta, q = np.zeros(N), np.zeros(N)
150     cash, total_internal_profit = 0.0, np.zeros(len(vertices))
151     loss_bound = N*b*np.log(2) # initial theta=0; R(z)=0 for every outcome
152     trades = [b*np.r_[np.zeros(3), np.full(3, np.log(.05/.95))],
153               b*np.array([.2, -.1, .05, -.3, -.2, -.4]),
154               b*np.array([-0.05, .1, -.15, -.2, -.4, -.2])]
155     records = []
156     for step, delta in enumerate(trades, 1):
157         payment = cost(theta+delta, b)-cost(theta, b)
158         cash += payment
159         q += delta
160         theta += delta
161         loss = vertices @ q-cash
162         assert max(loss) <= loss_bound+1e-9*b
163         old_theta = theta.copy()
164         theta, _, _, action = project_fw(old_theta, b, alpha)
165         internal_profit = (vertices @ (theta-old_theta)
166                           - cost(theta, b)+cost(old_theta, b))
167         assert min(internal_profit) >= -1e-9*b
168         total_internal_profit += internal_profit
169         # Internal corrections change neither real trader holdings nor cash.
170         hypothetical_base_loss = (vertices @ theta-cost(theta, b)
171                                   + cost(np.zeros(N), b))
172         assert
173         ↪ np.max(abs(loss-(hypothetical_base_loss-total_internal_profit))) <
174         ↪ 1e-9*b
175         assert max(loss) <= loss_bound-min(total_internal_profit)+1e-9*b
176         records.append(dict(trade=step, customer_payment=payment,
177                             actual_worst_loss=float(max(loss)), action=action,
178                             internal_worst_profit=float(min(internal_profit))))
179     return dict(alpha=alpha, original_loss_bound=loss_bound, actual_cash=cash,
180               actual_trader_positions=q.tolist(),
181               ↪ internal_state=theta.tolist(),
182               ↪ final_losses_by_outcome=loss.tolist(),
183               ↪ cumulative_internal_profit_min=float(min(total_internal_profit)),
184               ↪ tightened_loss_bound=loss_bound-float(min(total_internal_profit)),
185               ↪ trades=records)

```

Listing 16.9: Ledger test: real holdings and cash kept separate from the internal state

`ledger_test` keeps actual trader liabilities q and cash K separate from the internal state θ , as its docstring says; this is the separation of [Section 14.2](#). It starts from $\theta^0 = 0$, where $R(z) = 0$ for every outcome, so the loss bound is $Nb \log 2$ with $N = 6$ ([Equation \(7.1\)](#)). Three trades follow (lines 152–154). The first leaves singleton state coordinates at zero, hence singleton prices at $1/2$, and sets each conjunction state coordinate to $b \log(0.05/0.95) = b \logit(0.05)$, which moves the market from uniform prices to the starting prices p^0 of the triangle example.

For each trade (lines 157–162) the customer pays $C(\theta + \delta) - C(\theta)$ ([Equation \(2.1\)](#)); cash, holdings, and state all move; and the actual loss in every outcome, $q^\top z - K$ ([Equation \(14.2\)](#)), is asserted to be within the original bound. Then `project_fw` corrects the internal state (line 164). The internal trade’s profit is asserted to be nonnegative in every outcome, up to

tolerance, and is accumulated.

Independent check. Two invariants follow (lines 169–173). First, internal corrections change neither real trader holdings nor cash, so in every outcome the actual loss equals the loss $z^\top \theta - C(\theta) + C(0)$ implied by the internal state from $\theta^0 = 0$, minus the cumulative internal profit. Second, the worst actual loss is within the tightened bound: the original bound minus the worst-case cumulative internal profit. The returned dictionary reports both bounds, the cash, the positions, the internal state, and the final losses by outcome.

Status. `main` runs this test with $\min(\alpha, 0.95)$, so the default run used $\alpha = 0.95$. Its recorded result, a final worst actual loss below both the original and the tightened bound, is in Section 15.2.

16.11 The main program

`main` runs every check and is listed in three parts.

```

185 def main():
186     parser = argparse.ArgumentParser(description=__doc__)
187     parser.add_argument("--b", type=float, default=1.0)
188     parser.add_argument("--alpha", type=float, default=0.9999)
189     args = parser.parse_args()
190     b, alpha = args.b, args.alpha
191     if b <= 0 or not 0 < alpha < 1:
192         parser.error("require b > 0 and 0 < alpha < 1")
193     vertices = np.array([list(x) + [x[i]*x[j] for i, j in PAIRS]
194                         for x in itertools.product((0.0, 1.0), repeat=3)])
195     binary_vectors = np.array(list(itertools.product((0.0, 1.0), repeat=N)))
196     admitted = [z for z in binary_vectors if np.all(A @ z <= rhs+1e-12)]
197     assert {tuple(z) for z in admitted} == {tuple(z) for z in vertices}
198     p0 = np.array([0.5]*3 + [0.05]*3)
199     theta = state(p0, b)
200     # LP relaxation accepts p0, but the exact convex hull does not.
201     assert np.max(A @ p0-rhs) <= 1e-12
202     assert not membership(p0, vertices)
203     separator = np.array([-1.0]*3 + [1.0]*3)
204     lp = linprog(separator, A_ub=A, b_ub=rhs, bounds=(0, 1), method="highs")
205     _, ip_bound, ip_value = oracle(separator)
206     assert abs(ip_value - min(vertices @ separator)) < 1e-10
207     assert abs(lp.fun+1.5) < 1e-10 and abs(ip_value+1) < 1e-10

```

Listing 16.10: `main`, part 1: arguments, payoff audit, and the relaxation–hull gap

`main` parses `--b` and `--alpha` and rejects $b \leq 0$ or $\alpha \notin (0, 1)$. This historical check is incomplete for nonfinite liquidity; a general interface must require finite $b > 0$ explicitly. It enumerates the eight valid payoff vectors directly from the definition $y_{ij} = x_i x_j$ and audits the IP model: of the 64 binary six-vectors, exactly those eight satisfy $Az \leq \text{rhs}$ (line 197). The starting prices p^0 give $\theta = b \logit(p^0)$. The LP relaxation accepts p^0 , but the exact convex hull does not (lines 200–202). Finally the separator objective $-\sum x + \sum y$, the direction of the initial arbitrage in Chapter 8, is minimized by LP over the relaxation and by the integer oracle. The assertions require the LP value -1.5 and the integer value -1 , the latter equal to the enumerated minimum.

```

208 # Enumeration reference: all eight vertices, numerical contraction 1e-12.
209 reference, _ = correct(vertices, vertices.mean(axis=0), 1e-12, theta, b)

```

```

211 # Symmetry and the active inequality 3a-3c=1 also give a closed form.
212 a = (1+np.sqrt(7))/9
213 analytic = np.array([a]*3 + [a-1/3]*3)
214 assert np.max(abs(reference-analytic)) < 2e-6
215 reference_value = divergence(analytic, theta, b)
216
217 # Central differences check the derivative used by the LMO.
218 test_mu = np.array([0.3, 0.6, 0.4, 0.2, 0.2, 0.2])
219 step, eye = 1e-6, np.eye(N)
220 fd = np.array([(divergence(test_mu+step*e, theta, b)
221                 - divergence(test_mu-step*e, theta, b))/(2*step) for e in
222                 ↪ eye])
223 gradient_error = float(np.max(abs(fd-(state(test_mu, b)-theta))))
224 assert gradient_error < 1e-6*b
225
226 final_theta, best, trace, action = project_fw(theta, b, alpha)
227 final_prices = expit(final_theta/b)
228 delta = final_theta-theta
229 profits = vertices @ delta - (cost(final_theta, b)-cost(theta, b))
230 f = divergence(final_prices, theta, b)
231 g_exact = float(delta @ final_prices - min(vertices @ delta))
232 assert membership(final_prices, vertices)
233 assert abs(min(profits)-(f-g_exact)) < 1e-10*b
234 assert min(profits) >= -1e-10*b
235 if action == "safe update: accuracy threshold":
236     assert min(profits) >= alpha*reference_value-1e-9*b
237 # Bernoulli negative entropy is 4b-strongly convex in Euclidean norm.
238 assert np.linalg.norm(final_prices-analytic) <=
239     ↪ np.sqrt(best["gap_upper"]/(2*b))+1e-6

```

Listing 16.11: main, part 2: reference projections, gradient check, and the certificate checked against every outcome

Two references for the projection come next. The enumerated reference runs `correct` over all eight vertices with contraction 10^{-12} ; the analytic reference is $\mu^* = (a, a, a, a - 1/3, a - 1/3, a - 1/3)$ from Equation (15.1). (The code comment writes the active inequality as $3a - 3c = 1$, with c for the conjunction price called r in Section 15.1.) The two must agree to 2×10^{-6} , and the optimal arbitrage value $F_{\theta^0}(\mu^*)$ is evaluated at the analytic point. Central differences with step 10^{-6} then check the derivative used by the oracle. The test point need only lie in the strict binary box; coherence is unnecessary for this unconstrained derivative check. The hardened tests use several interior points and step sizes and independently check the weight-space chain rule.

The main projection follows. Its result is checked directly against outcomes: `profits` holds the correction's profit in each of the eight outcomes, and `g_exact` is the exact gap g computed by enumeration. The assertions require that

- the final prices are coherent;
- the worst outcome profit equals $F_{\theta^0}(\mu) - g$ (Equation (10.1)) to within $10^{-10}b$;
- the worst outcome profit is nonnegative, to within $10^{-10}b$;
- in the successful main fixture, the historical `accuracy threshold` label accompanies a profit at least α times the optimal value; the same label alone does not establish this for the distinct tiny-objective branch;
- the final prices lie within $\sqrt{\bar{g}/(2b)} + 10^{-6}$ of the analytic projection; Bernoulli negative

entropy is $4b$ -strongly convex in the Euclidean norm.

A coherent candidate's gap, strong convexity, and constrained optimality give

$$2b\|\mu - \mu^\star\|_2^2 \leq F_\theta(\mu) - F_\theta(\mu^\star) \leq \bar{g}.$$

The fixture adds 10^{-6} for numerical error. This bounds the default's saved candidate, not an unchanged state with different prices.

```

239     # A coherent target alone is insufficient to preserve the loss guarantee.
240     unsafe_prices = vertices.mean(axis=0)
241     unsafe_theta = state(unsafe_prices, b)
242     unsafe_delta = unsafe_theta - theta
243     unsafe_profit = float(min(vertices @ unsafe_delta
244                               - cost(unsafe_theta, b) + cost(theta, b))
245     assert membership(unsafe_prices, vertices) and unsafe_profit < 0
246     ledger = ledger_test(vertices, b, min(alpha, .95))
247     early_theta, _, _, early_action = project_fw(theta, b, alpha,
248     ↪ max_iterations=1)
249     late_theta, _, _, late_action = project_fw(theta, b, alpha,
250     ↪ max_iterations=7)
251     assert np.array_equal(early_theta, theta)
252     late_profit = float(min(vertices @ (late_theta - theta))
253     ↪ - cost(late_theta, b) + cost(theta, b))
254     assert late_profit > 0 and membership(expt(theta/b), vertices)
255     bound_audit_error = max(lb - float(min(vertices @ c)) for c, lb in
256     ↪ ORACLE_AUDIT)
257     assert bound_audit_error <= 1e-10*b
258     output = dict(
259         environment=dict(python=platform.python_version(),
260         ↪ numpy=np.__version__,
261         ↪ scipy=scipy.__version__), b=b, alpha=alpha,
262         security_order=["x1", "x2", "x3", "y12", "y23", "y13"],
263         ip_structural_vectors_checked=len(binary_vectors),
264         ip_structural_valid_vectors=len(admitted),
265         initial_prices=p0.tolist(), initial_local_constraint_max_violation=0.0,
266         initial_exact_hull_membership=False,
267         triangle_inequality_violation=float(sum(p0[:3]) - sum(p0[3:])-1),
268         lp_oracle_value=float(lp.fun), ip_oracle_value=ip_value,
269         ip_oracle_lower_bound=ip_bound, lp_oracle_solution=lp.x.tolist(),
270         enumerated_reference_prices=reference.tolist(),
271         analytic_reference_prices=analytic.tolist(),
272         optimal_arbitrage_profit=reference_value,
273         ↪ gradient_max_error=gradient_error,
274         ↪ fw_action=action, fw_final_prices=final_prices.tolist(),
275         ↪ fw_best_iteration=best["iteration"], fw_iterations=len(trace),
276         ↪ fw_divergence=f, fw_exact_gap=g_exact,
277         ↪ fw_profit_lower_bound=best["profit_lower_bound"],
278         ↪ fw_worst_outcome_profit=float(min(profits)),
279         ↪ fw_all_outcome_profits=profits.tolist(),
280         ↪ coherent_unsafe_target=unsafe_prices.tolist(),
281         ↪ coherent_unsafe_update_worst_profit=unsafe_profit,
282         ↪ oracle_calls_audited=len(ORACLE_AUDIT),
283         ↪ max_oracle_lower_bound_minus_exact_min=bound_audit_error,
284         ↪ ledger=ledger, interruption_tests=dict(early_action=early_action,
285         ↪ late_action=late_action, late_worst_outcome_profit=late_profit),
286         ↪ fw_trace=trace)
287     print(json.dumps(output, indent=2))

```

```

284
285 if __name__ == "__main__":
286     main()

```

Listing 16.12: `main`, part 3: unsafe reset, ledger and interruption tests, oracle audit, and the JSON output

The remaining checks start from the comment that a coherent target alone is insufficient to preserve the loss guarantee. Moving to the marginals of independent fair bits, $(0.5, 0.5, 0.5, 0.25, 0.25, 0.25)$, which are coherent, is asserted to have a negative worst-case profit (lines 239–245); compare [Section 10.3](#). The ledger test runs next (line 246). The interruption tests call `project_fw` with budgets of one and seven iterations: one iteration must return the original state; seven must return a state with positive worst-case profit and coherent prices (lines 247–252). Finally, every lower bound recorded in `ORACLE_AUDIT` is compared with the exact enumerated minimum, and none may exceed it by more than $10^{-10}b$ (lines 253–254). Only after every assertion has passed does `main` assemble and print the JSON object.

The all-outcome identity and nonnegative-profit checks test mathematical invariants, within their stated numerical tolerances. The one- and seven-iteration expectations are regression fixtures for this implementation. A correct alternative can reach its first safe candidate at another iteration. A general return API should test safe update, unchanged return, small objective, no candidate, and deadline cases separately.

16.12 Reading the JSON output

Read the action before interpreting a certificate. For the historical successful `main` fixture, the returned state is the best accepted candidate: its objective is approximately 0.06763091685, its enumerated gap is 0.00000302942, and the worst outcome profit is their difference, 0.06762788743. The guarded lower certificate is 0.06762711998. These are respectively an objective value, an optimality residual, an all-outcome profit calculation, and a conservative numerical certificate.

Other returns from the historical `project_fw` need more care. It can return the original state while `best` still describes a rejected candidate. The fields `fw_final_prices` and `fw_divergence`, together with the enumerated profits, describe the returned state. The searched candidate supplies `fw_best_iteration` and `fw_profit_lower_bound`. The returned state and searched candidate coincide in the successful fixture but can differ after an unchanged return. An unchanged trade has zero profit; a rejected candidate’s negative certificate must not be attached to it. A generalized schema should place candidate diagnostics and returned-state diagnostics in separate records and include an explicit acceptance flag. The hardened module supplies separate candidate and returned-state diagnostics for the triangle ([Section 16.13](#)).

16.12.1 Historical output schema

The JSON object has the keys of [Table 16.1](#), in the order `main` writes them; [Table 16.2](#) lists the keys inside `ledger`, `interruption_tests`, and `fw_trace`. Two entries, the maximum local-constraint violation and the exact-hull membership flag of the initial prices, are constants written once the assertion that establishes them has passed. The recorded values of the default run are in [Table 15.1](#); they are historical.

Table 16.1: Top-level keys of the JSON object printed by `main`.

Key	Content
<code>environment</code>	Python, NumPy, and SciPy versions of the interpreter that ran the script
<code>b</code> , <code>alpha</code>	The parameters b and α from the command line
<code>security_order</code>	Coordinate order: $x_1, x_2, x_3, y_{12}, y_{23}, y_{13}$
<code>ip_structural_vectors_checked</code>	Number of binary six-vectors tested against $Az \leq \text{rhs}$
<code>ip_structural_valid_vectors</code>	Number admitted; asserted to be exactly the eight valid payoff vectors
<code>initial_prices</code>	p^0
<code>initial_local_constraint_max_violation</code>	Written as 0.0 after the assertion that p^0 satisfies every local row
<code>initial_exact_hull_membership</code>	Historical key name; false after numerical LP failure, with nonmembership independently proved by the triangle inequality
<code>triangle_inequality_violation</code>	Sum of the singleton prices minus the sum of the conjunction prices, minus one
<code>lp_oracle_value</code>	LP-relaxation minimum of $-\sum x + \sum y$
<code>ip_oracle_value</code>	Integer-oracle incumbent value for the same objective
<code>ip_oracle_lower_bound</code>	The guarded lower bound m_{lb} returned with it
<code>lp_oracle_solution</code>	The LP minimizer
<code>enumerated_reference_prices</code>	Projection over all eight vertices with contraction 10^{-12}
<code>analytic_reference_prices</code>	μ^* from Equation (15.1)
<code>optimal_arbitrage_profit</code>	$F_{\theta^0}(\mu^*)$ at the analytic point
<code>gradient_max_error</code>	Largest central-difference error of $\nabla R(\mu) - \theta$ at the test point
<code>fw_action</code>	<code>safe</code> <code>update</code> or <code>unchanged</code> , followed by the stop reason
<code>fw_final_prices</code>	Prices of the returned state
<code>fw_best_iteration</code>	Iteration with the best certificate
<code>fw_iterations</code>	Number of Frank–Wolfe iterations run
<code>fw_divergence</code>	$F_{\theta^0}(\mu)$ at the final prices
<code>fw_exact_gap</code>	g , computed by enumeration
<code>fw_profit_lower_bound</code>	Best candidate’s guarded certificate; it describes the returned state only when that candidate is accepted
<code>fw_worst_outcome_profit</code>	Worst correction profit over the eight outcomes; asserted equal to $F_{\theta^0}(\mu) - g$
<code>fw_all_outcome_profits</code>	The eight outcome profits
<code>coherent_unsafe_target</code>	The coherent reset target (0.5, 0.5, 0.5, 0.25, 0.25, 0.25)
<code>coherent_unsafe_update_worst_profit</code>	Worst-case profit of moving to that target; asserted negative
<code>oracle_calls_audited</code>	Number of oracle calls whose bounds were audited by enumeration
<code>max_oracle_lower_bound_minus_exact_min</code>	Largest excess of a returned lower bound over the exact minimum; asserted at most $10^{-10}b$
<code>ledger</code>	The ledger test (Table 16.2)
<code>interruption_tests</code>	The one- and seven-iteration runs (Table 16.2)
<code>fw_trace</code>	One record per Frank–Wolfe iteration (Table 16.2)

Table 16.2: Keys nested inside `ledger`, `interruption_tests`, and the records of `ledger.trades` and `fw_trace`.

Key	Content
<i>In ledger</i>	
<code>alpha</code>	Extraction parameter used, $\min(\alpha, 0.95)$
<code>original_loss_bound</code>	$Nb \log 2$, the loss budget from $\theta^0 = 0$
<code>actual_cash</code>	Cash K collected from the three trades
<code>actual_trader_positions</code>	Real holdings q
<code>internal_state</code>	Internal pricing state θ after the last correction
<code>final_losses_by_outcome</code>	$q^\top z - K$ for each of the eight outcomes
<code>cumulative_internal_profit_min</code>	Worst-case cumulative profit of the internal corrections
<code>tightened_loss_bound</code>	Original bound minus that profit
<code>trades</code>	One record per trade, with the five keys below
<i>In each record of ledger.trades</i>	
<code>trade</code>	Trade number, 1 to 3
<code>customer_payment</code>	$C(\theta + \delta) - C(\theta)$
<code>actual_worst_loss</code>	Largest $q^\top z - K$ over outcomes after the trade
<code>action</code>	Action returned by <code>project_fw</code> for the correction
<code>internal_worst_profit</code>	Worst outcome profit of that correction
<i>In interruption_tests</i>	
<code>early_action</code>	Action returned with a one-iteration budget
<code>late_action</code>	Action returned with a seven-iteration budget
<code>late_worst_outcome_profit</code>	Worst outcome profit of the seven-iteration state
<i>In each record of fw_trace</i>	
<code>iteration</code>	Iteration number
<code>active_vertices</code>	Size of the active set when the iteration starts
<code>contraction</code>	Current contraction ε
<code>divergence</code>	$F_\theta(\mu)$ at the iterate
<code>gap_upper</code>	Computed guarded gap; an upper bound when the oracle lower bound is valid
<code>gap_incumbent</code>	Gap computed from the oracle's incumbent value
<code>profit_lower_bound</code>	Certificate $F_\theta(\mu) - \bar{g}$

16.13 Versioned hardening and revision checks

The historical listing remains byte-identical. The separate `fwmm_triangle_hardened.py` adds input validation, explicit numerical membership states, checked oracle bounds, and separate action and stopping-reason fields. Its tests live in `test_fwmm_triangle_hardened.py`. For this eight-outcome model, the variant audits numerical oracle results against all payoffs before using a certificate. That finite numerical audit supplies an additional check for this instance; it is not a proof that an arbitrary MIP solver produces exact bounds.

Version 2.1 checks the inverse state directly: a finite extreme state can round to an endpoint price without being its finite inverse. It rejects nonfinite intermediate gaps and profit bounds, and validates the status, primal point, and recomputed objective of a separate LP reference. That reference is a numerical comparison, not a profit certificate. Acceptance requires a nonnegative lower profit bound after subtracting the explicit objective and gap evaluation allowances.

This variant is still synchronous and specific to the six-coordinate triangle. Its oracle requires a completed solve with both a validated payoff and bound. The independently optional witness/bound fields, versioned settlement, deadlines, and atomic engine commits in [Chapter 19](#) are proposed interfaces, not capabilities of this module. Numerical failure raises an explicit exception; an integrating engine must preserve its own accounts.

```
python -m unittest discover -s code -p "test_fwmm_triangle_hardened.py"
python code/fwmm_triangle_hardened.py
node tools/verify_revision.mjs
python tools/verify_followup.py
```

A separate new run of the unchanged snapshot used Python 3.10.9, NumPy 2.2.6, and SciPy 1.15.3. Its output is `revision/snapshot-new-run.json`. It is distinct from the historical numbers printed in [Chapter 15](#). The `revision manifest` records source hashes, execution results, test status, and build checks. The independent JavaScript audit checks selected arithmetic and rejection cases; the Python tests exercise the new module. Neither is a replay of the NCAA experiment. The follow-up `execution record` separates version 2.1 runs, optimized-mode tests, and deliberate invalid-input diagnostics from the earlier version 2.0 records. The historical program and the review reconstruction have different bytes but identical parsed program structure; [the comparison record](#) identifies the omitted comments and printing-related indentation differences.

Exercises.

1. From a newly recorded `results.json`, identify the original state, searched candidate, returned state, outcome set, and oracle bound for one certificate. Verify the subtraction giving its worst profit and compare the candidate iteration with the final iteration.
2. Reconstruct one ledger row using its pre-trade state. Verify that the internal correction leaves cash and outside holdings unchanged and that its hypothetical-profit term gives [Equation \(14.2\)](#).
3. Give an invalid oracle lower bound that makes the raw upper gap negative. Explain why clamping it to zero fails, and specify a rejection test.

Part VI

Evidence

Chapter 17

The original NCAA experiments

Separate the original paper’s historical replay results from deployment evidence, scalability claims, and experiments still to be reproduced.

The paper replays reconstructed orders from a 2010 market through three alternative engines. With its reported 30-minute projection limit, the first full projection completed only after 45 games had settled. Some earlier gains came from logical settlement. These results concern the reported reconstruction and implementation, rather than traders responding to a live FWMM deployment.

Provenance. The results below are those reported in arXiv:1606.02825v2, 10 June 2016 [P1] and described in the original notes dated 12 October 2026. This editorial revision does not claim a new reproduction or refresh the source-availability search.

17.1 Data, reconstruction, and engine settings

Paper. The evaluation uses historical bets from Yahoo’s Predictalot market for the 2010 NCAA men’s basketball tournament. The pipeline is recorded bets, reconstructed limit orders and budgets, an inferred initial state and settlement timeline, engine runs, and forecast evaluation [P1, §§5.1–5.2].  §§5.1–5.2, pp. 14–16

Item	What the paper reports
<i>Data selection</i>	
Tournament	64 teams, 63 games, 2^{63} complete outcomes
Original recorded bets	93,036
Selected subset described in §5.1	63,689 bets, about 68% of the original data
Comparison bets in the selected subset	About 17%
Supported market language	Atomic team/game events, comparisons of team wins, and comparisons of conference-win sums
<i>Reconstructed inputs and engine settings</i>	
Treatments	Independent markets (IND), LCMM, and LCMM plus Frank–Wolfe projection (FWMM)
Base liquidity	$b = 150$
Per-order budgets	0.1, 1, 10, 100, and 1,000
Randomized replay inputs	Three seeds for the inferred limit prices
Historical implementation	Java and Gurobi 5.5
Projection time limit	30 minutes
<i>Observed runtime milestone</i>	
First completed full projection	March 21, 2010, 13:58:50, after 45 games had settled

Caution. The introduction also summarizes the experiment as roughly 88,000 trades on 5,000 securities. The detailed data section reports 63,689 selected historical bets. These are differently described counts; the manuscript does not provide a sufficiently explicit mapping to silently equate them. A reproduction should obtain the processed order stream and its schema, not infer one count from the other [P1, pp. 3, 14–15].

17.2 How orders were reconstructed

Paper. For each historical buy, the dataset contained purchased quantity and total cost. The authors compute the average price $\bar{p}$ and draw a limit price uniformly from $[\bar{p}, 1]$. The simulated order buys until its limit price is reached or its assigned budget is exhausted. Historical sells are transformed into purchases of a complementary bundle before that conversion.  §5.2, p. 15

For a hypothetical record, a purchase of two shares costing 0.6 has average execution price 0.3. A sampled limit in $[0.3, 1]$, together with a fresh assigned budget, defines its replay order. This example is not a historical observation. Each engine executes against its own changing prices, so the executed quantity can differ across engines even with the same input order.

Each order is treated as a new agent with a fresh fixed budget. This is not a model of persistent traders carrying inventories and updating their remaining budgets across many orders: no balance remains from that trader’s earlier order. The same randomized input files are used across the market treatments [P1, §5.2].

Interpretation. The replay is useful evidence about information propagation under a specified reconstruction of demand. It does not recover the traders' actual beliefs, establish how those traders would have reacted to different market histories, or demonstrate a trading strategy's realizable returns.

17.3 Initialization and settlement are reconstructed too

Paper. The original initial prices were unavailable. The authors derive initialization from observed prices in a six-hour window starting 27 hours and ending 21 hours before the first match, combine direct prices with tournament relationships, normalize, and perform an LCMM projection.

■ §5.1, pp. 14–15

Paper. The actual game-settlement times were also unavailable. Games are treated as settled 100 minutes after their recorded start times [P1, §§5.1–5.2].

These are proxy timestamps. An assumed settlement can precede the actual availability of a result; the timing protocol has not been validated here. A reproduction should use the actual time at which settlement information became available, or explicitly measure sensitivity to the timestamp assumption. Also align the initialization cutoff with the start of the replay to prevent unintended use of later information.

17.4 Forecast scores and reported results

Paper. For a categorical variable with realized outcome x^* , the paper evaluates $\log \mu_{j,x^*}$. For a purchased event bundle $X_j \in E$, it evaluates binary log likelihood, using the probability of the event or its complement according to what happened. In the paper's own symbols, the score of a bundle of the form $X \in \mathcal{E}$ is

■ §5.2, pp. 16–17

$$\mathbf{1}\{x^* \in \mathcal{E}\} \log \mu\{X \in \mathcal{E}\} + \mathbf{1}\{x^* \notin \mathcal{E}\} \log \mu\{X \notin \mathcal{E}\},$$

and a larger, less negative log likelihood indicates a better forecast [P1, §5.2, p. 16]. Equivalently, log loss is its negative and smaller is better.

Paper. The reported median improvements of FWMM over LCMM across budget settings and seeds are 3.3% for variable forecasts and 2.2% for purchased-bundle forecasts. Improvements become more substantial after projections begin completing. These are the paper's relative comparisons of its negative-valued log likelihoods, not percentage-point changes in classification accuracy [P1, §5.2 and Figs. 2–3]. The percentages are retained as reported; the notes do not establish their exact implementation denominator. A new reproduction should explicitly state the evaluation population, averaging weights, and sign convention. For example, $(L_{\text{LCMM}} - L_{\text{FWMM}})/L_{\text{LCMM}}$ for positive log losses is a possible relative-reduction convention, not an assertion that this was the historical computation.

The time-series score includes already settled variables. It therefore tends toward zero log loss as the tournament completes even without improving the prediction of still-uncertain events. The authors explicitly describe this effect. In a new experiment, report an unsettled-only score alongside any all-variable score and compare the same evaluation set across treatments. For example, two events with realized-outcome probabilities 1/2 each have mean log likelihood $-\log 2$. If one settles and receives probability one while the other forecast remains unchanged, the mean becomes $-\frac{1}{2} \log 2$. This improvement alone says nothing about

prediction of the remaining uncertain event. The proposed unsettled-only analysis must be reported separately from the paper's all-variable score.

17.5 What to conclude about scale

In this replay, the FWMM treatment improved the reported forecast scores once projections became computationally manageable; IP-based logical settlement also produced some earlier gains.

It does not demonstrate routine real-time full projections throughout the original 2⁶³-outcome market. The authors report that full projection succeeds only after substantial settlement has reduced the problem, with about five hours of engine time to replay trades originally spanning 22 days. This aggregate replay duration does not specify per-order latency. The paper also reports that the IP oracle dominates restricted-master optimization time. These are findings of that implementation and hardware, not guaranteed performance characteristics of every new model [P1, §§4.1, 5.2, 6].

■ §6, pp. 17–18

17.6 Code and data availability

Status. In the search reported by the original notes dated 12 October 2026, the notes did not locate an official public release of the original FWMM engine or original Predictalot transaction log in the sources checked: the supplied paper, the author's publication page, and targeted title/code/data searches. This does not establish that no release exists anywhere.

The original notes identified the author-associated Barrier Frank–Wolfe repository, which implements graphical-model inference, not the full FWMM market. Its README describes an older Python 2.7/Gurobi environment. Treat it as an algorithmic reference requiring porting and dependency review [S3]. The repository's identity and stated legacy requirements were checked in that investigation; the notes do not claim that it was installed or reproduced.

For a faithful historical reproduction, obtain the original engine or exact algorithm settings, security definitions, processed orders, initialization rules, trade-to-order transformation, random seeds, and settlement timeline. Public basketball scores alone cannot substitute for a transaction log. The synthetic experiment of Chapter 15, with the complete runnable code of Chapter 16, can be evaluated without those historical materials.

Exercises.

1. Design a settlement-time sensitivity study. Keep the reconstructed orders, initialization, seeds, engine settings, and evaluation population fixed; vary the delay from recorded game start to settlement. Specify an all-variable score, an unsettled-only score, and a runtime measure.
2. For the hypothetical two-share order, choose a limit price and budget. Explain why matched replay inputs need not produce equal executed quantities, and list the additional records needed to reproduce them.

Chapter 18

Public evidence of deployment

Assess separately the evidence for live operation, use of a specific mathematical method, and trading performance.

Status: as of 12 October 2026. The original notes report checking these sources on 12 October 2026. This editorial revision does not refresh that website or repository investigation. Live websites and default repository branches can change; the source-check date remains distinct from the revision date.

18.1 What the public evidence establishes

18.1.1 Context and finding

The construction investigated here uses a sum of logarithmic market scoring rule (LMSR) costs, Bregman projection onto coherent prices, Frank–Wolfe (FW), and an integer-programming (IP) oracle. The paper’s general framework starts from a bounded-loss cost; LMSR is the main instantiation discussed in these notes [P1].

The sources described in the original notes support three different kinds of connection:

1. Professional participants provide liquidity and use correlation-aware pricing and risk systems.
2. Integer optimization is explicitly described in an operational collateral application in the surrounding ecosystem.
3. A public Polymarket project was reported to contain a Frank–Wolfe-style loop and binary solver integration.

Key idea: the evidence limit. The sources reviewed do not establish that a named professional market maker on Kalshi or Polymarket runs the paper’s full LMSR/Bregman/Frank–Wolfe/IP construction in production. This is a limit of the available evidence, not a claim that no private firm uses it.

Totalis discloses a collateral-optimization application (Section 18.3); the notes’ Open-PolyTrader inspection addresses a public algorithm implementation (Section 18.4). They answer different questions.

18.1.2 How to interpret the evidence

Operation, method identification, and performance are independent claims. A firm can trade live without disclosing its optimizer; a repository can implement a method without live use; and operation alone does not establish profitability. The source-type labels below identify the evidence for each claim rather than assigning numerical confidence scores.

Table 18.1: Evidence categories: what each kind of source can establish.

Source type	What it can establish	What remains unresolved
Venue announcement or operator interview	A named firm's activity and broad workflow	Its private objective function, solver, and performance
First-party technical description of a running service	An explicit operational claim and disclosed method	Independent implementation and performance verification
Reported source-code inspection	What the reported inspected version implements	Whether that code is used live and whether it earns money
Published code and documentation	An available design and stated scope	Live operation and performance
Simulated or paper-trading study	Behavior under an experimental protocol	Real fills and live profitability
Reconstructed historical transactions	Retrospective activity under stated accounting assumptions	Private optimizer and prospective returns
Secondary reporting	An attributed report of activity	First-party confirmation and undisclosed methods
Official job posting	Intended workflow and hiring requirements	Completed deployment
Observed trading patterns	Transactions or behavior consistent with a strategy	The private program that generated those transactions

18.1.3 Evidence by organization or project

RFQ means *request for quote*; CLOB means *central limit order book*. A parlay pays according to a combination of underlying events.

Organization or project	Venue or setting	Evidence and qualification
White Swan Data	Kalshi and other RFQ platforms	Its founder describes live singles and parlay quoting, correlated exposure, and existing pricing/risk engines. No specific optimizer is disclosed. [E1]
Susquehanna / SIG	Kalshi	Kalshi announces SIG's institutional market-making desk. This establishes participation, without identifying FW/IP. [E2]
Wintermute	Kalshi and Polymarket, according to reporting	Decrypt reports both venues, attributing their names to an unnamed source. Algorithm details are absent. [E3]
DRW	Kalshi and Polymarket	An official job posting specifies dynamic quote skew, cross-platform arbitrage, and statistical fair-value models. A hiring description is weaker than a deployment report. [E4]
Totalis	Its own Solana parlay platform referencing Kalshi/Polymarket outcomes	The operator says its collateral engine uses HiGHS through SciPy MILP. This is portfolio collateral computation, with deployment self-reported. [E5; E6]
OpenPolyTrader	Experimental Polymarket trading system	The notes report inspected code contains FW-style iteration and OR-Tools CP-SAT. The objective and execution scope differ from a faithful FWMM implementation. [E7; E8; E9; E10; E11]
Polymarket's official maker bot	Polymarket CLOB	An experimental keeper offers Bands and AMM strategies. Publication of the bot does not establish a professional firm's use of it. [E12]
LayerX research bot	Polymarket	Its Bregman strategy is explicitly paper trading; live execution and market making appear as future work. [E14]

18.2 Evidence about professional market makers

The following sources address participation and trading workflow. Their verbs matter: a venue announces participation, a reporter attributes a claim, and a job posting describes intended work.

18.2.1 White Swan: the actual commercial problem

Evidence (Venue announcement or operator interview): **White Swan**. The July 2026 founder interview described in the original notes provides a firsthand account of correlation-aware parlay market making. Its founder describes live singles and parlay quoting on Kalshi and other RFQ platforms, correlated exposure, and existing pricing and risk engines. It supports the relevance of joint-event pricing and portfolio exposure to an operating business [E1].

The interview does not disclose integer programming, Frank–Wolfe, Bregman projection, or the paper's loss guarantee [E1].

A statistical correlation model and a logical feasibility model serve different purposes. Correlation affects estimated probabilities. Logical feasibility excludes impossible settlement

combinations. The interview establishes the former business requirement; it does not identify the implementation of the latter.

18.2.2 Susquehanna / SIG: venue-confirmed participation

Evidence (Venue announcement or operator interview): **Susquehanna / SIG**. Kalshi’s 3 April 2024 announcement identifies SIG’s institutional market-making desk. This establishes participation on Kalshi. It does not identify a Frank–Wolfe method, integer-programming oracle, or private pricing objective [E2].

The announcement is therefore stronger evidence of institutional market-making activity than of any particular optimization technique.

18.2.3 Wintermute: reported venues, undisclosed algorithm

Evidence (Secondary reporting): **Wintermute**. Decrypt’s 29 May 2026 report identifies Kalshi and Polymarket as venues where Wintermute provides liquidity. The venue names are attributed to an unnamed source. Algorithm details are absent [E3].

This is secondary reporting with a specific attribution limit. It should not be restated as a first-party disclosure that Wintermute uses FWMM on both venues.

18.2.4 DRW: intended strategies in an official posting

Evidence (Official job posting): **DRW**. DRW’s official Prediction Markets Trader posting names Kalshi and Polymarket and specifies dynamic quote skew, cross-platform arbitrage, and statistical fair-value models [E4].

Those terms identify a relevant trading workflow. A hiring description is weaker than a deployment report: it does not establish that a particular optimization system has been completed, deployed, or independently evaluated. Nor does it identify the FWMM construction.

18.2.5 How to use these findings

These cases justify studying joint-event pricing, inventory, and risk as commercial problems. They do not determine which objective or solver to reproduce. A venue partnership can establish commercial participation while leaving both mathematics and profitability undisclosed.

18.3 Totalis, integer optimization, and exact payoff models

18.3.1 The disclosed application and its limits

Evidence (First-party technical description of a running service): **Totalis**. Totalis says it recomputes worst-case portfolio exposure with an integer linear program (ILP) whenever a maker's book changes, using HiGHS through SciPy MILP. Its article explicitly separates collateral computation from pricing and correlation estimation. Deployment is self-reported [E5].

Its documentation places quoting and settlement on **Totalis's own Solana parlay platform**, with underlying outcomes drawn from Kalshi/Polymarket. This is an application on a separate platform, rather than an operator disclosure by a named maker on the underlying venues [E5; E6].

The displayed article date precedes an interview it cites, so the notes treat it as content retrieved on 12 October 2026 rather than assigning every statement to its displayed original publication date.

18.3.2 One feasible-state model, different jobs

The following is an interpretation of the relationship between the research and the disclosed applications. It is not a claim about an undisclosed firm's implementation.

Let $\phi(\omega)$ be the vector of contract payoffs under settlement state ω , and let

$$\mathcal{Z} = \{\phi(\omega) : \omega \text{ satisfies the contract rules}\}.$$

The reusable component is optimization of a linear function over $\mathcal{Z}$. The objective and surrounding system change between applications.

Worst-case liability

For a liability vector q , compute

$$W(q) = \max_{z \in \mathcal{Z}} q^\top z.$$

This asks which feasible settlement would produce the largest payout obligation. Cash already received, collateral already locked, and venue-specific accounting must be handled separately. An internal risk estimate does not by itself change the collateral an exchange requires.

A Frank–Wolfe direction

For a differentiable convex objective F and current point μ , compute

$$z^* = \arg \min_{z \in \mathcal{Z}} \nabla F(\mu)^\top z.$$

This selects a feasible payoff vector minimizing the current linearization of the objective. It does not select a vertex by the largest finite decrease of the nonlinear objective. Frank–Wolfe then uses vertices or mixtures of vertices to update its iterate. The paper's particular objective, cost-function construction, and accepted-update conditions matter to its guarantees [P1]. In the paper's construction $F = F_\theta$, and this is the oracle problem (9.1) with $c = \nabla F_\theta(\mu)$.

An executable trading portfolio

A trading strategy must additionally select order quantities and sides under the prices, depth, inventory, fees, and execution rules it actually faces.

A coherent probability estimate can guide quoting, but coherence alone does not create positive expected profit. Likewise, an inconsistency in displayed midpoints does not prove that the corresponding portfolio can be purchased at a positive guaranteed margin.

These three problems can share a feasible-state model while requiring different objectives, bookkeeping, and validation. In particular, liability is a maximum of $q^\top z$, whereas the oracle is a minimum of $\nabla F(\mu)^\top z$. Fees, collateral rules, and cost-function charges are additional application-specific accounting terms; a shared feasible set does not make their guarantees interchangeable.

18.3.3 Encode payoffs for every intended objective

For binary leg indicators $\ell_1, \dots, \ell_k$, a parlay payout indicator satisfies

$$y = \ell_1 \wedge \dots \wedge \ell_k.$$

An exact linear encoding is

$$y \leq \ell_i \quad \text{for all } i, \quad y \geq \sum_i \ell_i - k + 1, \quad y \in \{0, 1\}.$$

If an objective only maximizes positive liabilities, upper bounds can sometimes suffice to obtain the correct maximum value: the objective pushes a payable ticket's indicator upward. That shortcut does not generally encode the full payoff set for arbitrary signed objectives.

For example, fix two binary legs at $\ell_1 = \ell_2 = 1$. With only $0 \leq y \leq \ell_1$ and $y \leq \ell_2$, minimizing y returns zero even though the parlay pays one. The missing inequality $y \geq \ell_1 + \ell_2 - 1$ excludes that false payoff. The failure concerns the feasible set itself, not numerical solver accuracy.

A Frank–Wolfe oracle receives changing gradient coefficients, potentially of either sign. Reusing an encoding designed only for positive liability maximization therefore requires care. **Test the feasible assignments themselves, not just one maximization result.**

Also distinguish **logical impossibility** from **unlikely correlation**. Removing an unlikely but valid state can understate worst-case exposure.

Chapters 8 and 9 give deeper coverage of the outcome oracle and the Frank–Wolfe method.

18.4 Public implementations and their mathematical limits

The cases below concern a reported static inspection, a published quoting baseline, and a paper-trading experiment. Each supports a different claim.

18.4.1 OpenPolyTrader: the notes' reported code inspection

Evidence (Reported source-code inspection): **OpenPolyTrader**. The author's research preview explicitly cites Kroer et al. [E7]. The original notes report inspecting these files:

File	Observed role
src/agents/projection/fw/FwLoopEngine.ts	Calls an oracle using objective gradients, maintains vertices, and invokes optimization over their convex combinations. [E9]
src/agents/projection/fw/objective.ts	Defines a quadratic penalty around an interior point minus a linear edge reward. [E10]
services/ip-oracle/app.py	Builds binary variables and linear constraints, then calls OR-Tools CP-SAT. [E11]

For the version described in those notes, with regularization value one, the reported objective is

$$f(x) = \frac{1}{2} \|x - u\|_2^2 - e^\top x,$$

where u is the supplied interior point and e contains edge coefficients (called c in the source). This is an experimental quadratic optimization adaptation; the inspected objective is **not the paper's LMSR-aligned entropy projection** [E9; E10].

The README as described in the original notes identifies Polymarket as active, Kalshi as scaffolded, and FW basket execution as selected YES/NO pair purchases. That investigation found no independently verified fills or profit record [E8].

Certificate detail: the notes report that the inspected loop accepts an oracle status of either optimal or feasible. Its stopping calculation uses the returned assignment, and it also permits stopping for a small iterate change. The loop does not use the oracle's returned global bound. Consequently, a convergence flag from this implementation should not be treated as the paper's mathematical certificate [E9; E11].

This is an attributed historical inspection report. The cited URLs point to mutable branches; neither their retrieval date nor this revision identifies an immutable code snapshot. No new repository audit or trading trial is claimed here.

A reproducible new audit must record repository, commit SHA, file path, line range, and relevant configuration, and link or archive that revision. Until then, the findings above remain reports about the original investigation. Coefficient conversion and global-bound rescaling would be inspection tasks, not defects established by the supplied evidence.

18.4.2 Why solver bounds matter

For gradient $c = \nabla F(\mu)$ and feasible incumbent $\hat{z}$, let $m_{\text{inc}} = c^\top \hat{z}$. The original-domain gap of Equation (9.2) satisfies

$$c^\top \mu - m_{\text{inc}} \leq g.$$

Thus a small value computed from a merely feasible oracle solution can understate the true gap. If the minimization solver also provides a valid global lower bound m_{lb} , then

$$g \leq c^\top \mu - m_{\text{lb}}.$$

This upper bound, the upper gap $\bar{g}$ of Equation (11.1), is suitable for a conservative stopping check. Record feasibility, incumbent objective, global bound, runtime, and stopping reason separately. A timeout and a proof of optimality are different experimental outcomes. Chapter 11 derives the bound contract and its safe-return conditions. A small iterate change does not supply the missing global lower bound.

18.4.3 Polymarket’s official maker bot: a quoting baseline

Evidence (Published source code and documentation): **Polymarket’s official maker bot.** Polymarket’s experimental keeper maintains orders around the midpoint using **Bands** or **AMM** strategies. The AMM documentation describes emulating concentrated liquidity. These sources disclose an actual quoting design, but no joint-outcome IP oracle or FWMM probability projection. Publication of the bot does not establish a professional firm’s use of it [E12; E13].

The official venue documentation separately describes inventory-sensitive quoting and order maintenance. It also says batched orders are evaluated individually. That execution behavior matters when testing a multi-leg strategy; a batch interface alone does not imply atomic portfolio execution [E15].

18.4.4 LayerX: explicitly paper trading

Evidence (Simulated or paper-trading study): **LayerX research bot.** In the source described by the original notes, the LayerX research bot’s Bregman strategy is explicitly paper trading. Live execution and market making appear as future work. It is evidence of experimentation, without demonstrating deployment of that strategy as a live market maker [E14].

18.5 Arbitrage profits and the limits of deployment evidence

Historical transactions can reveal activity and support a reconstructed profit estimate. They do not identify a private objective or solver, or establish that the trader supplied passive liquidity.

18.5.1 What the frequently repeated “\$40 million” result means

Evidence (Reconstructed historical transactions): **Saguillo et al., AFT 2025.** The published AFT 2025 paper *Unravelling the Probabilistic Forest* reports approximately **\$39.59 million in estimated extracted arbitrage profits**. It reconstructs activity from historical transactions, including grouping an address’s executions within roughly one-hour windows. It does not identify traders’ FW/IP code; deeper strategy analysis is left open [P15, see especially §§7.1 and 7.4].

This is evidence that payoff inconsistencies were exploited. It cannot establish which private solver selected a trade, that the traders were professional market makers, or that today’s execution economics reproduce the historical estimate.

The distinction to retain in discussion is between a historical measurement of trading activity and a verified attribution of that activity to a particular algorithm. Neither a profit estimate nor an observed wallet supplies that attribution on its own.

18.5.2 What would strengthen the deployment claim

The strongest missing evidence would connect **operator, code or method, live activity, and accounting**:

1. A named market maker explicitly identifies the optimization objective and solver family used in live quoting.
2. A technical paper, talk, or implementation shows the same method.
3. Time-linked order/fill records or an independently assessed report connect that method to live activity.
4. Performance distinguishes spread capture, rewards, directional exposure, and arbitrage.

Exercises.

1. Choose one source in [Section 18.2](#). State its strongest supported claim about operation, method, and performance, and identify the additional evidence needed to establish a stronger claim.
2. Fix both legs of a two-leg parlay to one. Minimize its payout using the upper constraints alone, then include the lower constraint. Explain why a positive-liability maximization can conceal the missing constraint.

Part VII

Building further

Chapter 19

Implementation architecture

The engine processes real orders and computes internal price corrections. Real orders change trader holdings and cash. A correction changes only the pricing state; the engine accepts it after checking its certificate against the current market snapshot.

Status. This chapter proposes a research-engine architecture, not the original FWMM implementation [P1]. The six-security program in [Appendix C](#) and its separate hardened variant in [Chapter 16](#) provide small fixed-cost fixtures. The additional modules and protocols below remain proposed; the historical source-check date is in the source concordance.

19.1 Build the research engine in layers

Four immutable data objects connect the proposed modules. A `SecuritySpec` assigns each contract a stable name, payoff rule, and position in an ordered list of d security coordinates. A `MarketSnapshot` contains pricing state θ , cost specification and liquidity, accepted settlement information σ , outcome-model version, coordinate-order ID, and engine version. An `OracleResult` reports independently validated payoff and bound evidence. A `ProjectionResult` identifies its source snapshot, returned candidate, certificate, stopping witness, and proposed action. Settlement proposals use a separate `SettlementResult` containing proved facts and their source snapshot.

Only `engine.py` may commit changes. For a signed real trade δ , it atomically advances θ and aggregate outside holdings q by δ , adds the signed cost difference to net collected cash K , and increments the version. A correction worker reads a snapshot and returns values. If accepted, the engine changes θ and its virtual offset, leaving q and K unchanged. The account identity is $q = \theta - \theta^0 - v$ from [Section 14.2](#). Settlement is a third engine operation, with its own cost change and event record.

The following filenames are illustrative; they do not name supplied modules.

Module	Responsibility
<code>payoffs.py</code>	Ordered security specification and independent direct evaluation of $\phi(\omega)$
<code>outcome_model.py</code>	Integer formulation, payoff-coordinate projection, oracle and settlement evidence
<code>costs.py</code>	Cost, prices, conjugate, supported inverse map and conditioned costs
<code>projection.py</code>	Snapshot-only master, candidate records and certificates
<code>lcmm.py</code>	Corrections under a specified valid outer relaxation
<code>engine.py</code>	Atomic orders, settlements and corrections; accounts, versions and event log
<code>simulate.py</code>	Immutable worlds, signals and requested orders
<code>evaluate.py</code>	Forecast, arbitrage, computational and accounting measurements

The direct payoff evaluator implements contract rules independently of the integer-constraint builder. A solver assignment may include auxiliary variables; an explicit projection extracts exactly the d security coordinates in the declared order. Auxiliary coordinates never enter entropy, prices, holdings or mixture vectors. Deduplicate by the exact decoded payoff vector, even when several auxiliary assignments produce it. Preserve a witness when needed to validate settlement compatibility or reuse a solver start.

19.1.1 Existing fixtures and proposed capabilities

The historical snapshot and the separately versioned hardened program cover the fixed six-security triangle. The following capability map separates what is present from engine features that still require implementation and tests.

Capability	Present in snapshot or hardened fixture	Proposed extension and required validation
Fixed triangle	Both supply payoff enumeration, completed oracle solves, mixture correction and a fixed-cost ledger.	General payoff models: exact set correspondence and certificate/account tests A1–A7 in Chapter 20 .
Numerical rejection	The hardened variant adds explicit finite-value, bound, mixture and inverse-price guards; the snapshot is unchanged.	Audit each new cost/model scale independently using A2–A6; toy checks are not a universal numerical guarantee.
Interrupted oracle	Both use an outer iteration cap; neither provides the general optional-evidence job interface.	Time/node/memory limits and witness-only/bound-only results: A2 and A8.
Logical settlement	Neither implements general INTFW or changing-cost settlement.	Proof-only implications, conditioned costs and retained liabilities: Stage D.
Atomic engine	Neither implements concurrent market commits or durable event replay.	Serialized engine first, then stale-job extension: A8 and the replay check in Section 19.6 .
Tournament pilot	Neither implements the four-treatment simulator.	Outcome/security checks in Stage C, then the saved-input pilot in Section 21.1 .

19.2 Define a precise oracle interface

An oracle request binds a finite coefficient vector $c \in \mathbb{R}^d$ to a snapshot, job, outcome model and coordinate order. Its objective ID hashes the coefficients in that order together with their numeric representation. The response echoes these IDs, minimization sense, and scaling convention. Matching the model version alone is insufficient: two objectives on the same constraints can have different lower bounds.

For the current compatible payoff set $\mathcal{Z}_\sigma$, the useful outputs are a validated payoff z and a finite bound $m_{\text{lb}} \leq \min_{z \in \mathcal{Z}_\sigma} c^\top z$, each optional. Recompute the incumbent objective $m_{\text{inc}} = c^\top z$ from the validated payoff. If the solver minimizes

$$\frac{c^\top z - \beta}{s}, \quad s > 0,$$

its lower bound l becomes $m_{\text{lb}} = sl + \beta$ in the original monetary units. Use $\beta = 0$ in the baseline, but log both s and β . Restore reported incumbent values by the same map and check them against recomputation. When both outputs exist, require $m_{\text{lb}} \leq m_{\text{inc}}$ with the declared numerical allowance. A relative MIP-gap field is not the Frank–Wolfe upper gap.

Implementation recommendation: oracle evidence and permitted actions. The candidate already has a valid mixture for this same snapshot.

Evidence	Permitted action
Payoff only	Add a column; it supplies no new lower profit certificate.
Finite lower bound only	Certify the existing candidate; invent no column.
Both	Add the column and evaluate the certificate independently.
Neither	Continue within budget or return unchanged with an explicit reason.
Proved empty payoff set	Report inconsistent model/settlement and block the affected operation.
Invalid or mismatched record	Reject it before computing a gap; never replace missing data by zero.

Validate a returned witness in this order:

1. Reject nonfinite entries and incorrect dimensions.
2. Check the distance to integers, then decode the rounded assignment.
3. Check variable bounds and every structural and settlement constraint. Integrality alone does not establish feasibility.
4. Extract the payoff coordinates and recompute the objective in its original units.

Log the residual and tolerance for each check. Before calculating a gap, also verify the record IDs and the finiteness of the bound. Reject contradictory evidence, including a materially negative upper gap. [Section 20.1](#) specifies the corresponding invalid-record fixtures.

Store raw solver status separately from normalized facts: validated witness available, valid bound available, infeasibility established, resource limit reached, or numerical failure. Time, node and memory limits are resource-limit subtypes. Solver-reported optimality uses its numerical tolerances, not exact arithmetic. A finite objective on a finite, nonempty binary

payoff set cannot be unbounded; an unbounded or unknown status is an interface/model failure, not evidence of small arbitrage. Record elapsed time and node count when available.

19.2.1 Later optimization

First implement the contract with a newly constructed solver model for every call. Persistent models and warm starts are optional performance changes with identical request/result semantics. Revalidate cached assignments under current settlement and recompute their objectives under current coefficients. Measure model construction and solve time separately before choosing caching.

19.3 Restricted-master choices

Let k active payoff vectors form the rows of $V \in \mathbb{R}^{k \times d}$. The anchor $u \in \text{conv}(\mathcal{Z}_\sigma)$ has its own feasible-mixture witness and is interior in every unsettled coordinate. The rows of $V_\varepsilon \in \mathbb{R}^{k \times d}$ are $(1 - \varepsilon)z + \varepsilon u$. For $\lambda \in \Delta_k = \{\lambda \geq 0 : \sum_j \lambda_j = 1\}$,

$$\mu = V_\varepsilon^\top \lambda \in \mathbb{R}^d, \quad c = \nabla \bar{R}(\mu) - \theta \in \mathbb{R}^d, \quad \nabla_\lambda F_\theta(\mu) = V_\varepsilon c \in \mathbb{R}^k.$$

The program's `weights @ contracted` is the row-vector equivalent. Here the current settlement is fixed and its subscripts are suppressed. Differentiate the smooth entropy extension on these feasible mixtures; the categorical simplex-domain indicator is not an ordinary differentiable function. The cost constant has zero derivative but remains in $F_\theta(\mu)$.

For a weight direction $h \in \mathbb{R}^k$, the Hessian-vector product is

$$H_\lambda h = V_\varepsilon D(\mu) V_\varepsilon^\top h, \quad D_{ii}(\mu) = \begin{cases} b/[\mu_i(1 - \mu_i)] & \text{reduced binary coordinates,} \\ b/\mu_i & \text{surviving categorical coordinates.} \end{cases}$$

It needs two matrix-vector products and one diagonal multiplication; its cost depends on the active matrix size. Fixed coordinates are excluded from these derivatives. Different weights can yield identical prices, so the weight Hessian need not be nonsingular and the weights need not be unique. Test this expression by directional finite differences as specified in [Section 20.1.3](#).

The inner solve uses only the active payoff set $\mathcal{Z}_t$. Its *restricted-master gap* is

$$g_{\varepsilon, \mathcal{Z}_t}(\mu) = c^\top \mu - \min_{z \in \mathcal{Z}_t} c^\top [(1 - \varepsilon)z + \varepsilon u]. \quad (19.1)$$

For a feasible active mixture, this nonnegative residual bounds objective suboptimality within that restricted master. It can vanish while columns outside $\mathcal{Z}_t$ still improve the objective. Keep it separate from the gap over the full contracted domain and from the current uncontracted-domain gap used for economic certification ([Section 12.4.1](#)). All three have monetary units. Apply an inner-solver accuracy target to a restricted-master residual, not to a gap that also measures missing columns.

An incomplete restricted solve can produce a usable candidate. Acceptance requires feasible mixture weights and a certificate evaluated at that actual candidate. Convergence and runtime for an inexact algorithm require separate analysis or experiments. The pilot uses the SLSQP policy specified in [Section 21.1](#). A proposed alternative uses projected gradient on the simplex. With $d_\lambda = V_\varepsilon c$, measure the residual $b\|\lambda - \Pi_{\Delta_k}(\lambda - d_\lambda/b)\|_\infty$. Start each

trial step at $1/b$ and halve it until the projected step satisfies Armijo decrease with coefficient 10^{-4} . Stop at residual $10^{-8}b$, after 100 steps, or at the job deadline. Record the residual and termination reason; a step limit conveys no optimality. Use the adaptive contraction of [Chapter 12](#), starting at $\varepsilon = 0.1$, and record every change. A finite-precision guard that prevents further contraction ends this proposed variant as numerically incomplete, with only its already certified fallback.

A `CandidateRecord` stores immutable references to V , λ , ε , the anchor and its witness, the snapshot, reconstructed μ , inverse state, oracle evidence and certificate. Preserve this whole record for the best candidate. After pruning small weights and renormalizing, reconstruct prices, inverse state, objective and oracle request, then recertify. A later contraction or column set must not reinterpret an earlier record. Exact payoff deduplication is separate from approximate equality of candidate prices.

19.4 Implement order execution from the cost function

For the binary-event order path, require $a \geq 0$ and $a^\top z \in \{0, 1\}$ for every current valid payoff. Initially support only a subset of one categorical block or one explicitly modeled event indicator. An unrelated sum of Yes securities can pay two dollars and fails this path; its limit must instead be defined as a monetary bundle price.

A buy request gives a finite budget $Q \geq 0$ in dollars, a marginal-probability limit $0 \leq \ell \leq 1$, and a finite share cap $s_{\max} \geq 0$. Reject nonfinite inputs and settled or constant directions. As an explicit policy, fill zero when $Q = 0$, the cap is zero, or the starting price is at or above the limit. Otherwise find the largest $s \in [0, s_{\max}]$ satisfying

$$C_\sigma(\theta + sa) - C_\sigma(\theta) \leq Q, \quad a^\top p_\sigma(\theta + sa) \leq \ell.$$

Convexity makes the directional marginal price nondecreasing. Check the cap first, then use a bracketed search on this finite interval for a binding budget or limit. Validate both inequalities after rounding the fill downward; a failed search returns zero with a reason. The equality convention at the starting limit is a policy choice, not a consequence of the weak inequalities.

For one unresolved binary LMSR with $b = 1$ and $\theta = 0$, take $Q = \log(3/2)$, $\ell = 3/4$, and $s_{\max} = 2$. Payment and ending price are

$$P(s) = \log(1 + e^s) - \log 2, \quad p(s) = \frac{e^s}{1 + e^s}.$$

The budget binds at $s = \log 2$, before the price limit at $s = \log 3$. The fill ends at price $2/3$ and leaves $\theta = q = \log 2$, $K = \log(3/2)$. Payment is the cost difference, not the fill multiplied by the starting price.

For a sale use the signed convention $\delta = -sa$, with a finite cap and a minimum marginal receipt $\ell_{\text{sell}} \in [0, 1]$. As in the cost-function model, signed holdings may be negative and traders must honor their settlement obligations. Require $a^\top p_\sigma(\theta - sa) \geq \ell_{\text{sell}}$, and fill zero if the starting receipt is at or below the limit. Update θ and q by $-sa$ and K by the signed cost difference. For example, from $\theta = q = K = 0$, selling $s = \log 2$ with cap $\log 2$ and limit $1/4$ gives

$$\theta = q = -\log 2, \quad \Delta K = \log(3/4) < 0, \quad p(\theta) = 1/3.$$

The maker pays $\log(4/3)$; the trader's settlement position is $-\log 2$ if the event occurs. A complement-based sale would need a separate complete-set/cash accounting derivation and is outside this baseline.

19.5 A minimal projection-job contract

Use a serialized first implementation: the engine queues new events while one settlement-and-correction operation is processed. A `SettlementResult` proposes only proved implications from its source snapshot. The engine checks and commits them through the prescribed settlement operation, then creates a new snapshot. Its q and K remain unchanged when no payout occurs. Freeze that new snapshot before any projection calculation.

Key idea: one mathematical problem per job. During a projection job, fix the accepted information σ . Every objective, inverse state, oracle bound and gap uses C_σ and $\mathcal{Z}_\sigma$. Only the restricted master uses a contraction of $\text{conv}(\mathcal{Z}_\sigma)$. The original-domain gap means the uncontracted domain of this current snapshot, not the pre-settlement outcome set. Numerical validation, a valid certificate, and an atomic current-snapshot check are three separate acceptance requirements.

One wall-clock deadline includes construction, implication search, anchor validation, masters and oracle calls. Pass the remaining allowance to solver interfaces that support it. Keep `max_iterations` as a separate guard. If a solver overruns or throws, record actual elapsed time and the phase; retain only previously validated evidence. This serialized proposal gives no low-latency guarantee. Zero time or iteration allowance skips projection and returns the exact input pricing state for the already installed cost snapshot.

Serialized operation (proposed).

```

read current snapshot; start the single job deadline
validate observations and proposed proved implications
if proved inconsistent: block operation; record the conflict
commit verified settlement; freeze its new cost/model snapshot
if all coordinates settled: return settlement-only, no master
validate payoff witnesses and an interior coherent anchor
if coverage unproved: return unchanged, initialization incomplete
while time remains and iteration count is below its cap:
    run restricted master; reconstruct and validate candidate
    request oracle evidence for this candidate and snapshot
    reject invalid evidence; use columns/bounds independently
    record certificate; preserve the best safe candidate
    record stop reason and its witness separately
    otherwise add columns and apply contraction policy
engine checks snapshot identity again, then commits atomically:
    correction changes pricing state and virtual offset only
    otherwise retain state for the installed settlement cost
record action, stop witness, returned candidate and failures

```

Initialization timeout establishes neither infeasibility nor a fixed value. Keep only proved facts and skip projection if an anchor lacks zero/one witness coverage for any unsettled binary coordinate. Never create an anchor by setting coordinates independently to one-half. Proved inconsistent observations block affected quoting and settlement installation pending reconciliation; an empty active list alone is not such a proof. If all coordinates are proved

fixed, the settled cost supplies their prices without an entropy master.

For valid candidate evidence, define $\bar{g} = c^\top \mu - m_{lb}$ and $\underline{\pi} = F_\theta(\mu) - \bar{g}$ in the frozen problem. A negative lower certificate is an ordinary uncertified candidate; retain a prior safe record or return unchanged. The best returned record maximizes the lower certificate, not minimizes the objective. Distinguish stop reasons: requested extraction established, small original arbitrage value established, budget exhausted, numerical failure, initialization incomplete, all settled, or stale snapshot. Store the stopping iteration separately from the returned candidate's iteration. For example, a witness with $\bar{g} \leq (1 - \alpha)F_\theta(\mu)$ establishes $\underline{\pi} \geq \alpha F_\theta(\mu) \geq \alpha \min F_\theta$. A stored candidate with a larger lower certificate inherits this bound; its own local ratio test need not have triggered stopping. A small objective upper bound permits an unchanged return without asserting coherent prices.

An asynchronous extension retains the same contracts. A job begun at version 7 cannot commit after an order creates version 8. New observations or a changed security mapping also invalidate it, even when the numerical state matches. Reuse old payoff witnesses only after current-model validation; discard old certificates. A settlement-only result records the new cost/version and unchanged real accounts. Workers never mutate live accounts while searching.

19.6 Metrics and replay records

A compact metrics row contains snapshot/job IDs, action, reason, requested budget, actual duration, iterations, solver status, active-set size, settled count, objective, bound, upper gap, certificate, and construction/master/oracle times. Record failures, inconsistent models, stale results, settlement-only and unchanged actions alongside accepted corrections. For orders, record requested direction, cap, budget/limit, fill, signed charge and before/after versions.

The replay record also stores the security specification, event stream, original and conditioned snapshots, candidate records, raw and normalized oracle results, configuration, code revision, Python/solver versions and thread counts. Include liquidity, extraction fraction, contraction policy, numerical margins, seeds and correction schedule. Store small arrays inline; store larger arrays as immutable artifacts with a relative path, schema and content hash. Supply those artifacts: a hash alone cannot reconstruct a candidate.

A replay check loads an accepted record and a rejected record in a fresh process, rebuilds their mixtures, prices, objectives and bound identities, and reapplies the snapshot and acceptance checks. For the version-7 rejection, the event stream must show the intervening version-8 order. This is an acceptance test for the proposed engine, not a capability claimed for the historical script.

Exercises.

1. Trace a correction started at version 7 after a real order creates version 8. Identify which candidate evidence may be revalidated and which certificate must be discarded.
2. Classify time-limited witness-only and bound-only results using the oracle action table. Which can add a column? Which can certify an existing candidate?
3. In the buy example, replace the budget by $\log 2$ and keep the price limit and cap. Compute the fill and identify its binding constraint.

4. Reconstruct a saved candidate after the active matrix and contraction change. List the immutable data needed to reproduce its original certificate.

Chapter 20

Validation experiments

A faster implementation is useful only after independent checks of its payoff model and account updates. The four validation stages below save test inputs, expected results, residuals and failure reasons.

Status. The triangle program supplies a small fixed-cost fixture. Stages A–D extend its tests into a proposed engine-validation suite; the suite is not a completed experimental result. The supplied code and separate hardening results are documented in [Chapter 16](#) and [Appendix C](#). They do not implement general settlement, asynchronous commits or tournament replay.

Each test uses one frozen mathematical snapshot: cost C_σ , state θ , security order and compatible payoff set $\mathcal{Z}_\sigma$. The contracts check payoff-model equivalence ([Chapter 8](#)), the correction-profit identity ([Equation \(10.1\)](#)), and real-account separation ([Section 14.2](#)) independently. Installing a new settlement starts a new snapshot; an old certificate cannot cross that boundary.

20.1 Stage A: algebraic and economic validation

20.1.1 Fixtures and tolerance registry

The triangle uses the ordered payoffs

$$(x_1, x_2, x_3, y_{12}, y_{23}, y_{13}), \quad y_{ij} = x_i x_j, \quad (x_1, x_2, x_3) \in \{0, 1\}^3.$$

This direct generator supplies eight valid payoff vectors. At the initial prices $p^0 = (0.5, 0.5, 0.5, 0.05, 0.05, 0.05)$, all pairwise AND inequalities pass, but

$$p_1^0 + p_2^0 + p_3^0 - p_{12}^0 - p_{23}^0 - p_{13}^0 = 1.5 - 0.15 = 1.35$$

violates the triangle bound by 0.35. Use $\theta_i = b \log[p_i/(1 - p_i)]$ for $0 < p_i < 1$. Save the numeric state and liquidity with each input.

Start with four named deterministic cases: `triangle` uses p^0 ; `coherent` uses the uniform-outcome mixture $(1/2, 1/2, 1/2, 1/4, 1/4, 1/4)$; `near_facet` uses $(a, a, a, r - 10^{-10}, r - 10^{-10}, r - 10^{-10})$, where $a = (1 + \sqrt{7})/9$ and $r = a - 1/3$; and `interior_extreme` uses $(10^{-8}, 1 - 10^{-8}, 1/2, 10^{-8}, 1/4, 10^{-8})$. The third case has cut violation $3 \cdot 10^{-10}$ and tests behavior near the numerical margin. Add 16 vectors sampled coordinatewise uniformly on $[0.05, 0.95]$ using NumPy PCG64 seed 20261014. Save their arrays, generator and version, so replay does not depend on regenerating them. Run each at $b \in \{0.1, 1, 10\}$.

Only `triangle` has the analytic target of Equation (15.1); perturbed inputs need their own reference.

The following proposed registry applies to these small double-precision fixtures. It is an engineering policy, not formal verification or a universal solver tolerance. Keep dimensionless and monetary residuals separate. Let $S = \max\{1, \|c\|_1/b, |F_\theta(\mu)|/b, |\Delta C|/b\}$ for the quantities present in a test.

Name	Proposed allowance	Residual and scope
<code>decode</code>	10^{-7} , dimensionless	Maximum distance to an integer before decoding. Decoded box/logic/set checks are exact.
<code>mix</code>	10^{-9} , dimensionless	Negative weight, sum-to-one and candidate reconstruction infinity residuals.
<code>price</code>	10^{-8} , dimensionless	Inverse-state price reconstruction infinity residual.
<code>money</code>	$10^{-8}bS$, dollars	Objective, bound ordering, profit and ledger residuals; record S for each comparison.
<code>deriv</code>	$10^{-6}b$ absolute; 10^{-6} relative	Directional derivative or Hessian-vector residual; retain both measurements.
<code>reference</code>	interval width $10^{-10}b$, dollars	Target uncertainty for an independently assessed all-vertex reference; unresolved if not attained.

A nonfinite field or wrong bound direction cannot be repaired by this registry. An allowance used to compare floating-point values does not turn an overstated bound into a proof. Record any conservative bound adjustment separately and require the resulting lower profit certificate to be nonnegative. A reference whose requested width is not attained is inconclusive, not a passing result.

20.1.2 Payoff and oracle contracts

A1: payoff-set equality. Enumerate all 64 binary six-vectors, retain those satisfying the integer formulation, and compare them exactly with the eight vectors from the direct generator. Check both inclusions after numerical assignments pass `decode` and their decoded bounds and constraints. Reject the model on the first extra or missing vector. Test two deliberately incorrect models. Removing $y_{12} \geq x_1 + x_2 - 1$ admits $(1, 1, 0, 0, 0, 0)$; for $c = (-1, -1, 0, 1, 0, 0)$, this vector has value -2 , below the valid minimum -1 . Adding $x_1 \leq 0$ instead excludes valid outcomes. The test must reject both models. Random objective tests alone cannot establish either inclusion.

A2: oracle evidence. Save coefficient vectors 0 , all signed unit vectors, the A1 vector, $(1, -2, 3, -4, 5, -6)$, and $(1, 1 + 10^{-10}, 0, 0, 0, 0)$, together with copies multiplied by 10^{-3} and 10^3 . Enumerate $m_\star = \min_z c^\top z$. A returned payoff must be valid; its recomputed objective must be at least $m_\star$ and a claimed lower bound must not exceed $m_\star$, using the recorded money allowance. Under solver-reported optimality also compare the incumbent to $m_\star$. Different tied minimizers pass. Check scaling with $s = 7, \beta = -3$: $l = (m_\star - \beta)/s$ must reconstruct $m_\star = sl + \beta$. Reject the wrapper on invalid evidence, even if its incumbent is optimal.

Use deterministic stubs to cover witness only, bound only, both and neither, with the actions in Section 19.2. Inject NaN, infinity, wrong vector length, an integral coordinate -1 , a violated auxiliary row, a violated settlement row, a stale objective ID on an unchanged model, and a mismatched snapshot/coordinate order. Reject each before certification. Also inject inconsistent bound/incumbent ordering, an unexpected unbounded status, and a materially

negative raw gap. Record exact rejection reasons; an implementation that clamps any of these into a zero gap must fail this test.

20.1.3 Derivative, mixture and objective contracts

A3: derivatives. For binary coordinates differentiate the smooth negative-entropy expression minus $\theta^\top \mu$ inside $(0, 1)^d$. For categorical blocks use the positive-coordinate entropy extension, or feasible tangent directions of the simplex; never take unrestricted differences of its domain indicator. Normalize each test direction to infinity norm one, use central differences with steps $h, h/10, h/100$, where $h = 10^{-4}$ unless the distance to a domain boundary requires a smaller initial step. Choose that smaller h below one tenth of the distance along the direction, so all evaluation points remain in the smooth domain. Save every step, absolute residual and relative residual; require two adjacent steps to meet `deriv`. Persistent failure rejects the derivative implementation.

For the master, use the eight payoff rows, their uniform anchor, $\varepsilon = 0.1$, and uniform positive weights. Test directions whose weight sum is zero. Compare finite differences to $V_\varepsilon c$ and the Hessian-vector product in [Section 19.3](#); perturbations must remain feasible. Near-boundary fixtures require smaller recorded steps and may remain unresolved because of cancellation. Do not loosen the registry silently.

A4: feasible candidate. Validate every active payoff against the current settlement. Check weights and the anchor's own mixture with `mix`, then reconstruct

$$\mu = (1 - \varepsilon)V^\top \lambda + \varepsilon u.$$

Require strict interior probabilities in unsettled coordinates separately from feasibility. Construct $\hat{\theta}$ with the supported entropy inverse map and verify $\nabla C_\sigma(\hat{\theta}) = \mu$ using `price`; only then set $c = \hat{\theta} - \theta$. Equivalently the pair must satisfy conjugate equality. A feasible candidate can pass with a large gap. Invalid/stale vertices, an unproved anchor or failed price reconstruction reject the candidate. After weight repair, reconstruct and recertify; a normalized weight vector alone is insufficient.

A5: objective reference. Compare the symmetric triangle to its analytic solution. For other fixtures solve over all enumerated vertices independently, with the same cost, liquidity and original state. Enumeration makes the domain exact but leaves a numerical convex problem. At each feasible candidate record

$$\max\{0, F_\theta(\mu) - \bar{g}\} \leq F_\star := \min_{v \in \text{conv}(\mathcal{Z}_\sigma)} F_\theta(v) \leq F_\theta(\mu).$$

Build the reference interval using an enumerated gap at its independently computed candidate, and seek reference width. Require the two intervals to overlap within money. Disjoint intervals reject the objective or certificate path; a wide reference blocks the claimed accuracy comparison. A successful optimizer status is not an exact-arithmetic objective reference.

20.1.4 Profit, accounts and interruption contracts

A6: profit identity. Store all eight values

$$\pi(z) = c^\top z - C_\sigma(\hat{\theta}) + C_\sigma(\theta).$$

Minimize these directly. Separately calculate $F_\theta(\mu)$ and $g = c^\top \mu - \min_z c^\top z$, then compare $\min_z \pi(z) = F_\theta(\mu) - g$ using `money`. For an inexact valid bound also check

$F_\theta(\mu) - \bar{g} \leq \min_z \pi(z)$. Removing the cost constant, reversing the cost difference or substituting an incumbent-derived gap must fail the certificate path.

A deterministic negative control uses the original triangle state and the coherent uniform-outcome candidate $\mu = (1/2, 1/2, 1/2, 1/4, 1/4, 1/4)$. Its direction is $(0, 0, 0, k, k, k)$ with $k = \log(19/3)$ and $b = 1$. Choose the valid but suboptimal incumbent $(1, 1, 0, 1, 0, 0)$ of value k . The exact minimum is zero, so the true gap is $3k/4$, while the incumbent estimate is $-k/4$. A deliberately incorrect rule that clamps that estimate to zero falsely accepts any extraction fraction below one because

$$F_\theta(\mu) = 3 \left[\frac{1}{4} \log 5 + \frac{3}{4} \log(15/19) \right] > 0, \quad \min_z \pi(z) = -3 \log(19/15) < 0.$$

Keep this broken rule outside the engine; the failure is independent of solver timing. The correct enumerated or valid-lower-bound rule rejects this reset.

A7: accounts and maintained bound. Start at the original triangle state with zero real accounts, attempt the unsafe coherent reset above, and compare q , K and every $q^\top z - K$ immediately before and after the attempted reset. They are unchanged even for the unsafe candidate, so this comparison alone cannot certify safety. Require the account identity, nonnegative correction certificate, and unchanged real accounts jointly. In a separate run from the same initial snapshot, apply the fixed real trades $0.1e_1, -0.05e_2, 0.02e_4$ and a certified correction after each trade. Record signed charges independently and check the maintained loss identity and original budget at every outcome with money. Reject the transition on any account or certificate failure. Negative hypothetical correction profit need not mean current actual loss has already exceeded the budget.

A8: interruption and atomicity. Deterministically interrupt before any candidate, during witness search, after a feasible candidate but before a useful bound, after a safe candidate, during a later solver call, and immediately before commit. Before a safe candidate exists, return unchanged; afterward a previously safe record may be used only if the snapshot still matches. Inject an order, an observed settlement and a coordinate-map change separately before commit; each forces a stale rejection. A thrown exception must leave no partial updates of θ , v , q or K . Verified settlement may already have committed as its own event; compare with that post-settlement snapshot and record a settlement-only action.

Use stubs for deterministic phase coverage and real solver deadlines for integration checks. Save requested budget, actual elapsed time, phase, status, action and exact before/after accounts. An outer iteration cap tests only outer truncation. It does not cover a time-limited MIP, initialization failure or a stale atomic commit. Reject the integration path if any rejected projection mutates the current snapshot or real accounts.

20.2 Stage B: compare local constraints with the exact payoff hull

The deterministic triangle diagnostic uses the same initial state and base cost in all rows. Define the pairwise outer relaxation by the binary box and, for each pair, $0 \leq y_{ij}$, $y_{ij} \leq x_i$, $y_{ij} \leq x_j$ and $y_{ij} \geq x_i + x_j - 1$. Its strengthened version also imposes $x_1 + x_2 + x_3 - y_{12} - y_{23} - y_{13} \leq 1$.

An outer-relaxation correction minimizes $F_\theta(\mu)$ over its stated domain, constructs the inverse state and checks the trade certificate on that domain. In an implementable dual form for $A\mu \geq a$, choose $\eta \geq 0$, set $\delta = A^\top \eta$, and require $a^\top \eta - [C(\theta + \delta) - C(\theta)] \geq 0$ after the declared numerical guard. This gives a lower profit bound on every valid payoff. Use the same deterministic coordinate-sweep policy specified for LCMM in [Section 21.1](#); save

its residual and budget. A feasibility LP alone specifies neither the correction trade nor its charge.

Diagnostic rule	Expected result at the triangle state
Independent costs, no correction	Retain initial prices; triangle violation 0.35.
Pairwise outer correction	Zero correction is optimal because the initial vector satisfies every outer inequality.
Pairwise plus triangle cut	Correct the violated facet; its exact optimum is the coherent analytic target in Chapter 15 . Check the returned state's own certificate.
Full payoff-hull correction	Obtain a coherent candidate without supplying that facet; report objective and gap, not an unsupported exact optimum.

Report both maximum outer-constraint violation and true-hull arbitrage. Validate numerical returns with A4–A6 and the registry. The one added cut suffices for this symmetric optimum; it does not describe every market's payoff hull. For a performance comparison use the same saved instance set and include constraint construction, solve and validation time. Measure certificate availability as well as oracle calls. Stronger formulations may take longer; that is a valid experimental result.

20.3 Stage C: small tournaments with enumeration

A fixed n -team binary single-elimination bracket with no draws has $n - 1$ games, each with two possible winners among its entrants. If either entrant can win each game, it has 2^{n-1} complete worlds: 8, 128 and 32,768 for 4, 8 and 16 teams. Distinct worlds can yield the same security payoff vector. Record both counts. Adding securities can distinguish more vectors without adding worlds. Enumeration memory also depends on the number of security columns and their stored numeric representation.

Save one outcome-by-security table per bracket, with named columns and an independent direct evaluator. Extend the language in order: game winners, exact team-win counts, comparisons of two teams' win counts, subgroup sums, and comparisons of sums. Check coordinate definitions, comparison orientation, equality cases and subgroup membership on every row. Reuse the sign-warning fixture from [Chapter 8](#): store the first mismatching world and security when a comparison sign is deliberately reversed. Removing one required logical constraint must admit a detected fictitious payoff.

For each language, prove the encoding correspondence structurally: every valid world extends to a feasible solver assignment, and each admitted assignment projects to an outcome-consistent payoff. Supplement it with exhaustive small bracket checks, both unconditioned and after saved observed-game restrictions. Compare decoded discrete payoffs exactly; use A2 and money for numerical objectives. Sampled signed-objective comparisons remain sampled tests. A mismatch rejects the model and blocks expansion to the next language or bracket size.

20.4 Stage D: partial settlement and logical propagation

INITFW searches for witnesses and proves fixed coordinates (Algorithm 3; [P1, Algorithm 3]). After each public observation in a valid chronological sequence, enumerate the surviving payoff set $\mathcal{Z}_\sigma$ and calculate

$$z_i^{\min} = \min_{z \in \mathcal{Z}_\sigma} z_i, \quad z_i^{\max} = \max_{z \in \mathcal{Z}_\sigma} z_i.$$

A coordinate is fixed exactly when $z_i^{\min} = z_i^{\max}$. Every value INITFW fixes must match this table. On a completed run require the same full set of fixed coordinates; after timeout require soundness only. Check exact decoded values, then require the new anchor's valid mixture and zero/one witness coverage in every unsettled binary coordinate. Missing coverage skips projection; a vector of independently assigned one-halves is not a substitute.

Run three distinct fixtures: a feasible model with interrupted witness search returns **unknown**; a feasible model with a proved implication records proved **fixed**; inconsistent observations with proved empty compatible set return **model inconsistent**. The last case blocks affected quoting and settlement installation pending reconciliation. An empty active list or solver limit establishes neither inconsistency nor no arbitrage. Revalidate all active payoffs after accepted settlement; A4–A6 must then use the new C_σ and $\mathcal{Z}_\sigma$, with only the master contracted.

20.4.1 Changing-cost accounts

For one binary block with $b = 1$, start at $\theta^0 = 0$ and buy $\log 2$ shares. As in Section 19.4, $\theta = q = \log 2$ and $K = \log(3/2)$. Observing that the event occurred changes the compatible set to $\{1\}$ and the cost from $C(t) = \log(1 + e^t)$ to $C_1(t) = t$. At the same state,

$$C(\theta) = \log 3, \quad C_1(\theta) = \log 2, \quad J = \log(3/2).$$

Installing the cost collects no cash and cancels no holding. The default test pays all obligations only at final settlement, so outstanding payment remains $\log 2$ and actual loss is $\log 2 - \log(3/2) = \log(4/3)$. An early-payment variant must record its cash transfer and discharged liability as a separate event.

For a nontrivial sequence test, append the six-security triangle as an independent unresolved component. Keep its coordinate order from Stage A and use $b = 1$ for all seven binary blocks. In the triangle component execute the signed trades $0.1e_1, -0.05e_2, 0.02e_4$ before the binary observation and $-0.04e_1, 0.03e_3, -0.01e_6$ after it. Attempt a certified correction after each trade, preserving unchanged returns too. Settle only the first, separate binary block. Enumerate the resulting $2 \cdot 8$ complete worlds before and eight after settlement. On a surviving outcome z , save each cost drop J_t and each outcome-specific internal correction profit $\pi_j(z)$. Independently verify the changing-cost identity

$$q^\top z - K = (\theta - \theta^0)^\top z - C_\sigma(\theta) + C(\theta^0) - \sum_t J_t - \sum_j \pi_j(z)$$

using money, as derived in Section 13.3. Check exact account immutability at a cost-only switch and fixed 0/1 prices under the settled cost. Erasing liabilities or counting J_t as received cash must fail. Reject settlement integration on any mismatch. Record runtime and certificate availability by fraction settled only after these checks pass.

20.5 Completion criteria for a first research prototype

The correctness milestone requires saved fixture inputs, independent expected results, residuals/tolerance entries, negative-control outcomes and failure logs for Stages A–D. It includes an enumerated 4- or 8-team bracket with verified atomic, sum and comparison securities. A test matrix states which engine paths have actually run and which remain proposed; passing the fixed-cost triangle does not complete it.

The separate pilot milestone compares IND, LCMM, LCMM_LOGICAL and FWMM under the canonical definitions in [Section 21.1.1](#) and the protocol in [Section 21.1](#). Observed settlement is common to all four; LCMM_LOGICAL adds proved logical implications to LCMM and omits full Frank–Wolfe correction. Save configuration, immutable common world/signal/request history, treatment results, versions, timings and all failure/unchanged actions. Prices and fills can differ across treatments. A result with no forecast improvement still completes a faithfully executed pilot. Neither milestone establishes scalability or live-execution readiness. Record larger instances that fail within budget before proposing a 64-team reproduction.

Exercises.

1. Remove the AND lower inequality in A1. Verify the fictitious payoff and signed objective values, and explain which inclusion of set equality fails.
2. For the negative control in A6, compute the eight direct profits. Can increasing a monetary tolerance repair the incumbent’s wrong bound direction? Contrast this with a 10^{-12} probability reconstruction residual.
3. Let a time-limited oracle supply a valid payoff but no bound. Identify which A2 actions may continue and which acceptance claim needs new evidence.
4. Apply the unsafe coherent reset in A6 without changing real accounts. Show why immediate actual loss is unchanged, yet the maintained-bound certificate rejects the reset. Then trace the binary settlement cash example.

Chapter 21

Simulation, evaluation, and venue data

First compare cost-function market makers on simulated outcomes and orders. Then study venue data with a separate model of executable prices and fills. The studies share a payoff model, but their pricing rules and profit claims need separate tests.

Status. This chapter specifies proposed experiments; no tournament simulator or venue replay has been run for this edition. The triangle programs in [Chapter 16](#) and [Appendix C](#) have a narrower scope. The pilot below fixes the generator, information, requests, initialization, and evaluation weights so that a first implementation has a reproducible target. Its numerical policies are proposed engineering choices, subject to the validation in [Chapter 20](#). Historical source checks remain dated 12 October 2026; targeted follow-up checks on 14 October 2026 are identified where used.

21.1 Stage E: a four-team pilot

The pilot asks whether full Frank–Wolfe correction improves forecast accuracy beyond LCMM with logical settlement, using the same generated histories and requests. Its primary summary is the paired difference in history-level categorical Brier scores between FWMM and LCMM_LOGICAL; [Section 21.3](#) defines the weights and comparison. Runtime, arbitrage certificates, and maker loss are separate outcomes. The small model may show no incremental forecast benefit, which remains a valid pilot result. The pilot is exploratory, with no significance claim.

Research proposal. Use teams A, B, C, D , semifinals A versus B and C versus D , and a final between their winners. Order the eight complete outcomes lexicographically by semifinal winners and final winner, and assign probability $1/8$ to each. Save this table and its payoff matrix. This joint law is equivalent to independent fair game draws conditional on the realized entrants. It never produces a final winner who lost a semifinal.

The security menu has seven categorical blocks: the winner of each of the three games and each team’s exact total number of wins in $\{0, 1, 2\}$. There are $2 + 2 + 4 + 4 \cdot 3 = 20$ indicator securities, each paying one dollar when its category occurs. The coordinate order is the two semifinal blocks, the final block, then win-count blocks for A, B, C, D ; categories use alphabetical or ascending numerical order. Comparisons and sums belong to the extension milestone below.

Common initial state. Use one categorical LMSR per block, all with $b = 1$ dollar. Let p^0 be the marginals of the uniform eight-outcome law and set $\theta_{j,x}^0 = b \log p_{j,x}^0$, with $q = 0$, $K = 0$, and $v = 0$. Thus semifinal prices are $(1/2, 1/2)$, final prices are $(1/4, 1/4, 1/4, 1/4)$, and every win-count block is $(1/2, 1/4, 1/4)$. The initial prices are coherent, $C(\theta^0) = 0$, and the initial loss budget is $10b \log 2$: each complete outcome contributes $2 \log 2 + \log 4 + 2 \log 4 + 2 \log 2$ to $-(\theta^0)^\top z/b$. There is no initial corrective trade. Model construction and witness searches still count in the reported initialization time.

Information and requests. Each independent history draws one hidden complete outcome. Public information reveals semifinal 1 after request 20, semifinal 2 after request 40, and the final after request 60. Before each request, choose uniformly from indicator coordinates that vary over outcomes compatible with public observations. For the chosen event e , a fresh trader receives $S \in \{0, 1\}$ with

$$\mathbb{P}(S = 1 \mid e = 1) = 3/4, \quad \mathbb{P}(S = 1 \mid e = 0) = 1/4.$$

Signals are independent conditional on the hidden outcome and chosen events. The generator uses that outcome to draw a signal; it never reveals the outcome or a future public observation to the trader. The trader sees only public observations, the selected event, and its private signal. Bayes' rule on the saved eight-outcome table gives its joint posterior and event belief $\ell = \mathbb{P}(e = 1 \mid \text{public information}, S)$.

Save all 60 request records before running a treatment. A record contains the event-time index, coordinate, buy side, marginal limit ℓ , fresh cash budget $Q = 0.1$ dollar, and quantity cap $s_{\max} = 1$ share. This pilot uses buys by fresh traders; it makes no claim about optimal portfolio choice. Under the budget/limit rule of [Chapter 19](#), choose the largest $s \in [0, s_{\max}]$ whose exact charge is at most Q and whose terminal marginal price is at most ℓ ; return zero if the initial quote is already at or above ℓ . Use bisection until the quantity interval is at most $10^{-10}b/(\$1 \text{ per share})$ shares, then take the feasible lower endpoint. Treatments receive identical requests, but their quantities, payments, and unfilled requests can differ. Persistent inventories and sells are separate experiments.

Use PCG64 with one saved seed per history (pilot seeds $1, \dots, 20$); consume first the outcome draw, then an event-selection draw and a signal draw for each request. Select a discrete item by the half-open cumulative-probability intervals in the stated order. Archive the draws and generated requests, generator/library version, and payoff matrix so replay does not depend on recreating a PRNG version.

21.1.1 Four treatments and their schedules

All treatments use the initialization, base costs, real-order rule, and public observations above. Direct observation fixes only the corresponding game-winner block through the cost restriction of [Chapter 13](#); inference about other blocks is an explicit treatment component. Public observations also restrict the current valid payoff set for every risk or certificate query.

ID	Additional settlement	Cross-block correction
IND	None	None
LCMM	None	Specified outer-constraint update below
LCMM_LOGICAL	Proved logical implications	Same LCMM update; no full Frank–Wolfe step
FWMM	Same logical search	Same LCMM update, then certified Frank–Wolfe job

After every request, including an unfilled one, install any scheduled public observation, run the applicable logical search, then the LCMM update, then the Frank–Wolfe job. Each stage sees the committed output of the preceding stage. Logical search runs only after a new public observation. It tests each remaining coordinate for both binary values in coordinate order, using the witness/proof contract of [Chapter 19](#); an inconclusive search leaves the coordinate unsettled. Search has a one-second elapsed-time allowance, shared across its queries. LCMM has 100 coordinate sweeps or one second, whichever is reached first. Frank–Wolfe has 100 outer iterations or one second, including initialization, and requests $\alpha = 0.9$ with $\varepsilon_F = 10^{-8}b$. An in-progress native call can overrun an allowance; record the overrun. All-settled states skip optimization and return that reason.

A specified LCMM update. For team t , write g_t for its semifinal-win marginal, f_t for its final-win marginal, and $n_{t,k}$ for its exact- k -wins marginal. Use

$$f_t = n_{t,2}, \quad g_t = n_{t,1} + n_{t,2} \quad (t = A, B, C, D).$$

Represent each equality by its positive and negative inequalities, ordered by team, equation, then sign, giving $A\mu \geq a$ with $a = 0$ before elimination of fixed coordinates. Substitute fixed coordinates into the right-hand side. Categorical normalization is enforced by the base cost. For this small menu these constraints may already describe the hull; the pilot permits zero incremental benefit from a full Frank–Wolfe correction.

Starting from $\eta = 0$, minimize

$$H(\eta) = C_\sigma(\theta + A^\top \eta) - a^\top \eta, \quad 0 \leq \eta_i \leq 20b.$$

One sweep minimizes each coordinate in the stated order, holding the rest fixed. Choose an endpoint when the derivative keeps one sign; otherwise bisect its monotone derivative to interval width $10^{-10}b$ and use the lower endpoint (the smaller coordinate in a flat interval). Stop when one sweep improves H by at most $10^{-10}b$, or a limit is reached. The finite multiplier cap specifies this pilot’s approximation, not the source LCMM algorithm. For $\delta = A^\top \eta$, validity of the inequalities gives

$$\min_{z \in Z_\sigma} \delta^\top z - [C_\sigma(\theta + \delta) - C_\sigma(\theta)] \geq a^\top \eta - [C_\sigma(\theta + \delta) - C_\sigma(\theta)].$$

Accept a nonzero update only if the finite, recomputed right-hand side exceeds the monetary guard defined below and the current snapshot still matches. Otherwise retain the state and record the reason. This margin is an engineering policy; a solver success flag alone cannot establish the inequality.

Frank–Wolfe and numerical reference. Use the current-settlement oracle, coherent anchor, and contraction procedure of [Chapters 12 and 19](#), with initial $\varepsilon = 0.1$ and floor 10^{-8} . Each job starts cold; witnesses use lexicographic payoff tie-breaking. Use validated lower bounds, original-domain gaps, and the same monetary acceptance guard. Failure to obtain witnesses or a certificate returns the current state with an explicit reason. No cached certificate is reused. The proposed baseline is Python 3.10.9, NumPy 2.2.6, SciPy 1.15.3, and its bundled HiGHS 1.8.0, matching the separately recorded local environment. Use SciPy SLSQP for the simplex-weight master with analytic objective/gradient, nonnegative bounds, the sum-to-one equality, `ftol=1e-12`, and `maxiter=2000`. Start the first master from uniform active weights and append zero weight for a new column. Validate mixture feasibility to 10^{-9} . Measure the restricted-master gap independently, with target $10^{-9}b$; a missed target remains visible even if SLSQP reports success.

Let μ be a feasible mixture of the contracted active atoms $(1 - \varepsilon)z + \varepsilon u$, $z \in \mathcal{Z}_t \subseteq \mathcal{Z}_\sigma$, with positive probabilities in every surviving categorical coordinate. Let c be its finite inverse-state trade. Define the restricted-master gap, full contracted-domain gap, and current original-domain gap, respectively:

$$\begin{aligned} g_{\varepsilon, \mathcal{Z}_t}(\mu) &= c^\top \mu - \min_{z \in \mathcal{Z}_t} c^\top [(1 - \varepsilon)z + \varepsilon u], \\ g_\varepsilon(\mu) &= c^\top \mu - \min_{z \in \mathcal{Z}_\sigma} c^\top [(1 - \varepsilon)z + \varepsilon u], \\ g(\mu) &= c^\top \mu - \min_{z \in \mathcal{Z}_\sigma} c^\top z. \end{aligned}$$

Only the first is the inner-master accuracy residual ([Equation \(19.1\)](#)). It can be zero while the second remains positive because columns are missing. Report the full contracted gap separately, and use only the original-domain gap, or its valid upper bound, in the profit certificate. When the oracle stops early, label bounds as bounds; the pilot’s separately timed enumeration audit can evaluate both full-domain gaps directly.

Call `scipy.optimize.milp` with binary payoff variables. Enable `presolve`, set `mip_rel_gap` to zero, and set `time_limit` to the remaining job allowance. In this pilot the integer model has one binary selector for each of the eight saved outcomes: $\xi_k \in \{0, 1\}$, $\sum_k \xi_k = 1$, and $z = Z\xi$, where columns of Z are the saved payoff vectors. Fix selectors for outcomes incompatible with current settlement to zero. Record native solver version, machine, all options and actual elapsed time. Decode integer values within 10^{-7} , then check the rounded payoff against every exact discrete constraint. Follow the dimensionless feasibility and inverse-price checks of [Chapter 20](#). In one query set

$$\tau_\S = 10^{-8}b \max\{1, \|c\|_1/b, |F|/b, |\Delta C|/b\}.$$

Subtract this engineering guard from the raw minimization lower bound and require a nonzero accepted correction’s recomputed lower profit certificate to exceed $\tau_\S$. Validate that guarded bound against all eight current payoffs in this pilot; record the raw discrepancy and reject inconsistent records. Enumeration here is a validation audit, separately timed. It is not

evidence that this guard suffices in a larger model. For the LCMM update use $c = \delta$, its displayed guaranteed-profit bound in place of F , and its cost difference in this scale rule.

An additional all-vertices numerical reference uses the same base costs, settlement policy, correction opportunities, and accounting as FWMM. It has all current payoff vertices available from the outset, a 60-second offline allowance, and an original-domain gap target $10^{-7}b$; halve the contraction when it prevents reaching that target, down to 10^{-12} . Use the same master and guard policies, replacing the integer solve by the enumerated minimum. Report the achieved interval when a target is missed (including when the guard exceeds the target). Exact domain enumeration does not remove numerical optimization error. A complete-outcome joint LMSR instead changes the regularizer and possibly subsidy; compare it as a separate market design (Section 9.10).

21.2 Measure four different outcomes

Computation. Measure each job from snapshot capture through returned result, including model setup, cache checks, witnesses, settlement work, oracle calls, master work, and validation. Also report one-time initialization and end-to-end history elapsed time. Break down measured phases, oracle calls and nodes, active-set size, and peak memory. Include failed jobs and deadline overruns. Medians and 95th percentiles describe jobs; uncertainty comes from independent histories. In an asynchronous extension, summed CPU phase times need not equal elapsed time. Fewer oracle calls alone do not establish faster execution.

Original-state arbitrage. Freeze accepted settlement information σ . Every objective and bound uses its matching C_σ and current uncontracted $\mathcal{Z}_\sigma$. Suppressing these subscripts within one query, define

$$A(\theta) = \min_{\mu \in \mathcal{M}} F_\theta(\mu), \quad \max\{0, F_\theta(\mu) - \bar{g}\} \leq A(\theta) \leq F_\theta(\mu).$$

The interval requires a feasible candidate, the stated inverse-state construction, and a valid original-domain upper gap. Without an oracle lower bound, retain a feasible objective as an upper bound and report only the universal zero-trade lower bound. Without a feasible candidate, the candidate upper bound is unknown. Record feasibility separately. The lower profit certificate describes a hypothetical correction; it is not cash collected. The interval bounds the original state's value, not residual arbitrage at an accepted coherent state.

At each event record values before the request, after its real fill, after public settlement, after accepted logical settlement, and after each correction action. Give every record its state/cost/outcome-set version. Recompute on a new snapshot when settlement changes. Keep unchanged and error returns and the number of requests since the last accepted correction. The pilot summarizes event time, so long jobs do not silently receive extra statistical weight; wall-time exposure is a separate scheduling measure.

Actual maker loss. For trader holdings q and net collected cash K , define $W(q) = \max_{z \in \mathcal{Z}_\sigma} q^\top z$. Actual worst loss is $W(q) - K$. Invoke the minimization oracle with $c = -q$. If its lower bound is m_{lb} and feasible incumbent objective is m_{inc} , then

$$-m_{inc} - K \leq W(q) - K \leq -m_{lb} - K.$$

Missing fields leave their endpoint unknown. A found payout scenario lower-bounds worst loss; the minimization lower bound supplies the conservative upper endpoint after sign reversal. Enumerate all eight outcomes in the pilot. Keep internal offsets, hypothetical correction profit, actual cash, and any venue-required collateral in separate fields.

Forecasts and information. Let $p_{j,x}(t)$ be the issued probability of category x of variable j at event time t , and let x_j^* be its realized category. Use

$$\text{NLL}_j(t) = -\log p_{j,x_j^*}(t), \quad \text{Brier}_j(t) = \sum_x [p_{j,x}(t) - \mathbf{1}\{x = x_j^*\}]^2.$$

This is the categorical-sum Brier convention. A binary probability 0.7 followed by outcome one has one-coordinate squared error 0.09 and categorical-sum score 0.18; do not average those conventions together without explicit weights.

The pilot's benchmark information $\mathcal{I}_t$ contains only public observations available at t , not private signals, the hidden outcome, or future observations. The evaluator computes $p_j^*(t) = \mathbb{P}(X_j = \cdot \mid \mathcal{I}_t)$ from the uniform table. It is a public-information comparison, not the posterior of an observer who also sees order flow. For a fixed forecast p_j , elementary expansion gives

$$\begin{aligned} \mathbb{E}_{X \sim p_j^*} [-\log p_{j,X}] &= H(p_j^*) + \text{KL}(p_j^* \| p_j), \\ \mathbb{E}_{X \sim p_j^*} \sum_x [p_{j,x} - \mathbf{1}\{X = x\}]^2 &= 1 - \|p_j^*\|_2^2 + \|p_j - p_j^*\|_2^2. \end{aligned}$$

The displayed identities hold for a fixed forecast and the stated public-information distribution. The market's forecast also depends on orders generated from private signals. To interpret the identities as conditional expected scores of that forecast, include its history in the conditioning information and recompute the outcome distribution. Comparisons with the public-information benchmark alone remain descriptive, and a single realized score does not identify the decomposition's separate terms. The forecast-error analysis of [P11] uses its own trader assumptions.

Save raw probabilities and stable log probabilities. A genuine zero assigned to the realized category produces infinite raw NLL; underflow is a numerical failure to distinguish and report. The optional displayed metric is *score-floored NLL*, $-\log \max(p_{j,x_j^*}, 10^{-12})$, with no renormalization. Report its mean alongside the raw infinite-event count and finite raw mean. Scoring adjustments never enter the market state or orders.

21.3 Replication and a common evaluation universe

At each of the 60 post-action event points, enumerate outcomes compatible with public observations. A variable is eligible exactly when at least two categories remain possible. This excludes every publicly determined variable even if a treatment has not discovered it. Use the same eligible variable IDs for every treatment, weight eligible variables equally within an event, then weight nonempty events equally within a history. The final all-settled event has no forecast score; record that fact instead of zero. Report logical-settlement discovery counts and delays separately.

The primary pilot summary S_h^T is that history's categorical Brier mean for treatment T . Secondary summaries are raw/score-floored NLL, post-trade and post-action arbitrage

intervals, total elapsed time, and maximum actual worst loss. For the primary comparison form

$$D_h = S_h^{\text{FWMM}} - S_h^{\text{LCMM_LOGICAL}}, \quad \bar{D} = H^{-1} \sum_{h=1}^H D_h, \quad \widehat{\text{se}}(\bar{D}) = s_D / \sqrt{H}.$$

Save each D_h . Histories are independent replicates; correlated snapshots inside one history are not extra replicates. The 20 histories are an exploratory pilot for debugging and variance estimation, with no significance claim. A practically relevant Brier difference is 0.01 under this convention; the target confirmatory half-width is 0.005. After the pilot, freeze all choices and choose $H_{\text{conf}} = \max\{100, \lceil (1.96s_D/0.005)^2 \rceil\}$ new histories, with seeds starting at 1001. This planning approximation does not guarantee the achieved width. Report the fixed-sample paired Student- t interval and its independent-history, finite-variance and approximate-normality assumptions. Do not keep extending that sample until a preferred result appears.

21.4 Plots and action categories

For every intended full-Frank–Wolfe correction opportunity, record one action: accepted coherent correction, unchanged-small-value, unchanged-insufficient-computation, or error. Record requested-extraction completion as a separate flag on accepted results; LCMM acceptance alone does not assert full coherence. A skipped all-settled opportunity is a separate category. Use all intended opportunities as the denominator, and display both accepted and completed-extraction fractions.

Plot these fractions against elapsed-time allowance, split by public settlement fraction. Plot oracle/master/total time in seconds against security count, and Brier/NLL against event time for exactly IND, LCMM, LCMM_LOGICAL, and FWMM. Show worst actual loss and its upper bound in dollars through each small replay. Every aggregate caption names history-level weighting and uncertainty method. A single correction trace of $F_\theta(\mu)$, $\bar{g}$, and $F_\theta(\mu) - \bar{g}$ against cumulative oracle seconds is a diagnostic, not a population estimate. Accepted-state coherence alone cannot describe how long live prices remain uncorrected.

21.5 Stage F: expand one factor at a time

The base pilot comprises 20 histories and four economic treatments: 80 runs, plus 20 offline numerical-reference replays. Its first extension varies only the Frank–Wolfe allowance to 0.1 and 10 seconds, adding 40 runs on the saved histories. Reuse the existing baseline runs. Changing an Frank–Wolfe contraction parameter does not create a new IND configuration.

Owner	Later factor	First comparison
Market	Teams and security menu	4 versus 8 teams; add comparisons after payoff tests
Trader process	Order-budget/liquidity ratio Q/b	0.01, 0.1, 1; save budgets and filled shares separately
Algorithm	Extraction, contraction, schedule	$\alpha = 0.5$ versus 0.9 at the same one-second allowance
Solver	Formulation and cuts	Same payoff language, equal certificate target, total elapsed time

Adaptive triggers, persistent traders, and warm starts are outside the first grid. A future cache is keyed by payoff-coordinate mapping and settlement/model version, filtered for current feasibility, and replenished by fresh witnesses; charge validation and replenishment time. A reused payoff never licenses reuse of an old bound. Predeclare at most the budget-by-market-size interaction after the pilot identifies feasible sizes; label other searches exploratory.

A strength-based extension must fix its prior and signal likelihood. One explicit outcome-law alternative draws s_t independently from $N(0, 1)$ at history start and runs the bracket with independent game-level uniforms and $\mathbb{P}(t \text{ beats } u \mid s) = \text{sigmoid}(s_t - s_u)$, where $\text{sigmoid}(x) = 1/(1 + e^{-x})$. Strengths remain evaluator-only unless a separate declared signal model reveals information about them.

Liquidity scaling check. For $k > 0$, categorical costs obey

$$C_{kb}(k\theta) = kC_b(\theta), \quad p_{kb}(k\theta) = p_b(\theta).$$

Charges, correction objectives, gaps, budgets, share quantities, and monetary tolerances scale by k under corresponding state/trade scaling. Test this scaling identity with $k = 10$: compare prices within the declared probability tolerance, and divide monetary outputs by k before comparing them. Keep probability tolerances, wall-time allowances, and event schedules unchanged. Matching price paths requires the same accepted decisions and appropriately scaled optimization policies; iterations, timings, and final digits remain measured outcomes.

21.6 Venue data requires an execution study

The remaining experiments use venue data and a separate execution model. They test the proposed methods; they do not reconstruct a private firm's strategy. The payoff model comes first, followed by liability, quote consistency, executable opportunity detection, and passive replay.

1. **Payoff rules.** Enumerate a small contract family and compare permitted payouts with the model. Record contract-rule versions and exceptions. Kalshi's combo documentation permits product settlement with scalar legs: $0.70 \cdot 1 \cdot 1 = 0.70$ [E18]. A binary AND encoding omits that payout and can give incorrect risk bounds. Include the scalar rule or justify a conservative representation; label a binary-only prototype's scope.
2. **Liability.** Compare joint worst payout with independent per-position bounds. Use the sign conversion above; distinguish model liability from venue-required collateral.
3. **Quotes.** Hold reference inputs, spreads, sizing, and inventory policy functions fixed while changing the consistency component. Report its residual and latency. Resulting fills and inventories may differ.
4. **Executable portfolios.** For a fixed filled vector δ with executable acquisition cost a_{exec} and modeled costs f , require

$$\min_{z \in \mathcal{Z}} \{\delta^\top z - a_{\text{exec}} - f\} \geq \gamma > 0$$

before calling it guaranteed arbitrage. In the diagnostic pilot set $\gamma = \$0.01$ per proposed basket and require full displayed depth at the decision snapshot, with fees charged by

archived rules. A correct detector may return no certified opportunity. Positive realized profit, profit conditional on full fill, and profit under all modeled fill scenarios are different outputs.

5. **Passive replay.** Report net realized P&L as the primary venue metric, with maximum modeled worst exposure, fill rate, and inventory as secondary measures. Use a declared 100 ms submission/cancellation latency, one-unit quotes, and cancellation/replacement at each input update. For order-level data use FIFO with all displayed size ahead at arrival; fills require subsequent traded volume to exhaust that queue. Ignore own market impact and label this assumption. With aggregated books, report two scenarios: no passive fills, and fills only after subsequent traded volume exceeds all displayed arrival depth at the price. A price touch alone never fills a quote. These are sensitivity models, not observations of hypothetical executions.

Two-leg execution check. Buy complementary one-dollar claims at 0.45 each with zero fees. Full fill guarantees 0.10; if only the first leg fills, cash falls by 0.45 and the worst settlement payoff is zero, so the minimum net payoff is -0.45 . Test rejection, delay, cancellation, and partial quantity on the second leg, recomputing cash and inventory after every actual fill. Polymarket's batch documentation states that orders are evaluated independently [E15]; successful submission is not evidence of a filled portfolio. The Kalshi and Polymarket passages were checked on 14 October 2026; this documentation check did not test live execution.

Reconstruct one decision. Retain immutable input data with hashes and availability permissions, exchange event time, local receipt time, and decision time. Save coordinate mappings, contract rules, cost/constraint/snapshot versions, executable depth, fees, slippage and latency assumptions, and pre-decision cash/inventory. Record objective sense and scale, incumbent, bound direction, status, and elapsed time. Keep proposed quantities separate from order acknowledgments, fills, cancels, and residual inventory after each lifecycle event. Store raw and net P&L in dollars, and liability bounds separately from required collateral. Replay may use only information received by decision time; a hash without retained data is insufficient. Correctness, added runtime, and absence of forecast or trading improvement are all reportable outcomes.

21.7 Implementation milestones

1. Preserve the triangle as the six-security regression fixture; pass its arithmetic, oracle, certificate, and ledger checks before changing the language.
2. Enumerate the four-team pilot, compare every payoff with the IP model, and verify common initial prices and loss budget.
3. Add sums and comparisons only after checking all ordered pairs, including the sign convention $Y = X_2 - X_1$.
4. Test observed and proved logical settlement, the conditioned cost, and the cash/liability identity across each cost change.

5. Run the four canonical treatments on saved paired pilot histories; preserve failed, skipped, and unchanged actions alongside accepted ones.
6. Scale and profile only after these gates pass. Save actual configuration count, timing boundary, bounds, and uncertainty plan before expanding.

Discussion questions.

1. Reconstruct a trader's posterior and saved request from one outcome, public-information state, and signal. Which fields can the trader see?
2. For one history, compute the paired Brier difference and the two-leg partial-fill loss by hand. Explain why they answer different questions.
3. As a separate research extension, replace the finite signal model or add comparison securities; state which pilot controls and validation gates change.

Chapter 22

Research extensions

Evaluate specific extensions to solver stopping, cut generation, active-set work, security design, and executable order-book arbitrage.

Status. The following extensions are research proposals, not implemented or measured results. The first five extend the cost-function maker; [Section 22.6](#) formulates a separate order-book problem with explicit executable prices and positions. The material was prepared on 12 October 2026 as part of a research guide; source availability and version statements reflect that research and were not rechecked.

22.1 Stop oracle search when the economic certificate is sufficient

Research proposal: stop on the economic certificate. Solving every pricing IP to complete optimality can be unnecessary. For a fixed candidate, its cost change $\Delta C = C(\hat{\theta}) - C(\theta)$ is known. A minimization lower bound satisfying

$$m_{\text{lb}} \geq \Delta C + \text{acceptance margin}$$

already certifies a safe correction (by [Equation \(11.2\)](#)). For the stronger fraction-of-optimal-profit condition ([Equation \(10.2\)](#) with [Equation \(11.1\)](#)), the threshold is

$$m_{\text{lb}} \geq c^\top \mu - (1 - \alpha)F_\theta(\mu).$$

Where the solver interface permits it, stop the oracle once the desired economic threshold has been proved, rather than waiting for an arbitrary relative MIP gap. Compare time-to-safe-update with time-to-optimal-oracle. This is a natural first implementation research question because the original paper reports oracle-dominated runtime ([Chapter 17](#)).

22.2 Turn oracle information into new LCMM cuts

Paper. The paper’s closing discussion observes that Frank–Wolfe can be used to construct separating hyperplanes that tighten the outer LCMM approximation, and so contribute to arbitrage removal even when there is no time to compute the projection [[P1](#), §6].  §6, p. 18

Research proposal: anticipated by the paper’s discussion. A valid lower bound $m_{lb} \leq \min_z c^\top z$ implies the valid inequality

$$c^\top \mu \geq m_{lb} \quad \text{for every } \mu \in \mathcal{M}.$$

If the current independent prices satisfy $c^\top p < m_{lb}$, this inequality separates them from the true hull. Add it to the outer LCMM system and reuse it for subsequent cheap corrections [P1, §6].

This suggests a combined row-and-column method: columns improve the current convex projection, while validated rows improve future outer relaxations. Use a bound, not merely an incumbent objective, on the right-hand side. A cut with an unjustified right-hand side can exclude real outcomes and destroy the intended guarantee.

Evaluate how many useful cuts are produced before a full projection would finish, whether they remain valuable across later trades, and how much extra work a growing LCMM system requires.

22.3 Spend more time on cheap active-set updates

Research proposal: cheap active-set updates. Cache feasible vertices, optimize weights over them, and request a new oracle call only when a new outcome or a new certificate is needed. Away-step, pairwise, and blended conditional-gradient ideas are relevant references [P10; P12].

Benchmark oracle calls and total time, not iteration counts alone. Keep the contraction and economic return rule. The convergence guarantees of smooth Frank–Wolfe variants do not automatically carry over to entropy on an uncontracted domain with unbounded curvature (Chapter 12).

22.4 Design the security language to fit the algorithm

Research proposal: security language. Compare generic Boolean conjunctions and comparisons with interval securities, sparse dependency graphs, tree-structured models, and bounded-width decompositions. Some structures admit specialized optimization or inference that can replace the generic IP oracle; some market languages admit much faster pricing mechanisms altogether [P13; P14; P16].

A particularly useful benchmark varies the density of pairwise conjunctions while holding the number of base events fixed. A dense graph and a tree can have similar-looking security definitions but very different global-consistency and optimization properties.

For a specialized mechanism, compare the complete design: supported contracts, exact versus approximate inference, loss assumptions, and measured computational cost. Do not assume a faster mechanism solves the same expressive market or uses the same entropy objective.

22.5 Separate information value from liquidity and compute

Research proposal: information, liquidity, and compute. The market can improve a log score because it corrects incoherence, propagates information, changes effective price impact, or receives more useful orders. Use the forecast-error literature to disentangle these effects [P11].

One focused study fixes the true data-generating distribution and trader signals, varies Q/b , and compares LCMM, logical settlement, and full projection at matched compute budgets. Another allocates extra correction time only when a preliminary bound suggests large arbitrage value. The objective is to estimate forecast improvement per unit of compute, with economic safety enforced throughout.

22.6 An order-book extension uses a different optimization problem

Research proposal: order-book formulation. If the eventual goal is to detect opportunities in observed exchange books, retain the payoff IP but replace the cost-function model with executable bids, asks, capacities, and fees. The AFT 2025 study is relevant context [P15].

For a simple single-price-level model, let $u_i \geq 0$ be units bought at ask a_i and $v_i \geq 0$ units sold at bid b_i^{bid} . (In this section u , v , a , and b^{bid} are local symbols, not the anchor, the virtual ledger, or the liquidity parameter of earlier chapters.) A worst-case payoff optimization can be written

$$\begin{aligned} \max_{u, v, \kappa} \quad & \kappa \\ \text{s.t.} \quad & (u - v)^\top z - a^\top u + (b^{\text{bid}})^\top v - \text{fees}(u, v) \geq \kappa, \quad \forall z \in \mathcal{Z}, \\ & 0 \leq u \leq \text{available ask depth}, \\ & 0 \leq v \leq \text{sellable inventory and bid depth}. \end{aligned}$$

For linear or suitably modeled piecewise-linear fees this can be handled with a master optimization plus separation over $\min_z (u - v)^\top z$. That is **constraint generation over worst-case outcomes**, distinct from the original FWMM's convex combination of outcome columns.

A snapshot certificate still assumes the specified fills can actually be obtained. Multiple order-book levels, timing differences, unfilled legs, settlement mismatches, and short-sale restrictions must be modeled before treating the result as executable profit. Projecting mid-prices or last-trade prices to a coherent vector does not by itself certify a tradable opportunity.

Discussion questions.

1. Which proposal in [Sections 22.1 to 22.5](#) best matches the bottleneck you expect to study, and what comparison would distinguish improvement from a changed market design?
2. For [Section 22.6](#), which real execution restriction should be modeled first: multiple price levels, inventory limits, leg fills, fees, or settlement matching?

Appendix A

Formula sheet and mistakes that change the method

Look up the candidate construction, profit certificate, and outside-account formulas with the assumptions needed to use them.

A.1 Construct a candidate in one fixed problem

Fix the starting pricing state θ , accepted settlement information σ , current cost C_σ , and nonempty compatible payoff set $\mathcal{Z}_\sigma$. Throughout one job, every objective, inverse state, oracle bound, and gap uses this same problem. Below, C , $R = C^*$, $\mathcal{Z}$, and $\mathcal{M} = \text{conv } \mathcal{Z}$ suppress its settlement subscripts; Ω denotes the compatible worlds. *Original domain* means this current uncontracted set, not the initial pre-settlement set. Only the restricted master uses a contraction of $\mathcal{M}$.

The candidate $\mu \in \mathcal{M}$ must have a validated convex-mixture representation. Its finite inverse state $\hat{\theta}$ must satisfy conjugate equality, $R(\mu) = \hat{\theta}^\top \mu - C(\hat{\theta})$. For reduced binary LMSR this requires $0 < \mu_i < 1$ on every unsettled coordinate and gives $\hat{\theta}_i = b \log[\mu_i/(1 - \mu_i)]$, where $b > 0$. For categorical LMSR, every surviving category must have positive mass in its normalized block; fixed coordinates use the prescribed settled cost. The entropy extension $\bar{R}$ supplies the inverse on those active coordinates. See the categorical inverse construction in [Section 10.1](#) and conditioned costs in [Section 13.2](#); an arbitrary conjugate need not admit this construction ([Section 6.7](#)).

Object	Formula in the fixed problem	Derivation
Valid payoff vectors	$\mathcal{Z} = \{\phi(\omega) : \omega \in \Omega\}$	Equation (3.1)
Coherent price set	$\mathcal{M} = \text{conv } \mathcal{Z}, \quad \mu \in \mathcal{M}$	Chapter 3
Charge for a real signed order δ	$C(\theta + \delta) - C(\theta)$	Equation (2.1)
Instantaneous prices	$p(\theta) = \nabla C(\theta)$	Equation (2.2)
Correction objective	$F_\theta(\mu) = R(\mu) + C(\theta) - \theta^\top \mu$	Equation (4.3)
Candidate state	$\hat{\theta} = \nabla \bar{R}(\mu)$	Section 10.1
Candidate trade and oracle coefficients	$c = \hat{\theta} - \theta$	Section 10.2
Integer-oracle minimum	$m^* = \min_{z \in \mathcal{Z}} c^\top z$	Equation (9.1)

Here z is one valid payoff vector, whereas μ is a coherent mean of such vectors. The instantaneous price is a derivative; finite orders pay the cost difference in the table. A finite entropy value at an endpoint does not give a finite derivative there.

A.2 Certify the trade and interpret the return

Before the following tests, validate finite inputs and intermediate results, candidate feasibility, the inverse-state identity, and the bound's direction, objective, units, and state/cost/outcome identifiers. Require $m_{\text{lb}} \leq m^\star$ for this coefficient vector and current payoff set. A materially negative upper gap fails validation; replacing it by zero does not repair the evidence. A live engine must also check that the snapshot is still current when it commits the correction. The frozen-state commit contract is in [Chapter 19](#), and the negative controls are in [Section 20.1](#).

Write $F^\star = \min_{v \in \mathcal{M}} F_\theta(v)$ for the original state's optimal arbitrage value under the tutorial's assumptions. Define the hypothetical trader's worst profit for the particular trade c by

$$\Pi = \min_{z \in \mathcal{Z}} [c^\top z - C(\hat{\theta}) + C(\theta)].$$

The inverse-state equality above gives the two equivalent certificate forms below. Neither hypothetical profit nor its lower certificate is real cash collected by the maker.

Quantity or test	Formula	Derivation
Exact original-domain Frank–Wolfe gap	$g = c^\top \mu - \min_{z \in \mathcal{Z}} c^\top z$	Equation (9.2)
Upper gap from a valid lower bound	$\bar{g} = c^\top \mu - m_{\text{lb}} \geq g$	Equation (11.1)
Worst hypothetical correction profit	$\Pi = F_\theta(\mu) - g$	Equation (10.1)
Lower profit certificate	$\underline{\Pi} = F_\theta(\mu) - \bar{g}$	Equation (11.2)
	$= m_{\text{lb}} - [C(\hat{\theta}) - C(\theta)]$	
Mathematical safe-return test	$\underline{\Pi} \geq 0$	Section 11.1
Fractional extraction test	$\bar{g} \leq (1 - \alpha)F_\theta(\mu), \quad 0 < \alpha < 1$	Equation (10.2)
Small original arbitrage value	$F_\theta(\mu) \leq \varepsilon_F, \quad \varepsilon_F \geq 0$	Chapter 10

The exact Frank–Wolfe gap bounds objective error: $0 \leq F_\theta(\mu) - F^\star \leq g \leq \bar{g}$. It differs from both objective error itself and a solver's relative MIP gap. After validating the inputs, numerical acceptance uses the declared monetary margin $\tau_\Pi \geq 0$, requiring a validated finite lower certificate at least τ_Π . Any allowance for evaluating the certificate must already be included in that lower bound; choosing a margin alone does not validate the calculation.

The fractional test establishes hypothetical profit at least $\alpha F^\star$. An accepted feasible candidate has coherent prices even when it extracts less than the optimum. The forgone profit is not residual arbitrage at those prices. Separately, a feasible candidate with $F_\theta(\mu) \leq \varepsilon_F$ permits an unchanged return labeled *small original arbitrage value*. It neither certifies that candidate's state change as safe nor bounds coordinatewise price error. Zero tolerance is meaningful mathematically; a practical finite-termination tolerance is normally positive.

A.3 Keep outside accounts separate

For these rows, initialize real outside holdings, virtual offsets, and real cash at $q_0 = v_0 = K_0 = 0$, and keep the security coordinates fixed. The reference convention leaves claims outstanding until terminal payout. Thus q records unpaid outside holdings, K is net cash collected from real trades, and v accumulates internal pricing corrections. A virtual correction changes θ and v together without changing q or K .

Account or loss	Formula	Derivation
Real trader holdings	$q = \theta - \theta^0 - v$	Equation (14.1)
Maker loss at one payoff z	$L(z) = q^\top z - K$	Equation (14.2)
Worst maker loss over current outcomes	$\max_{z \in \mathcal{Z}} (q^\top z - K)$	Chapter 7

Installing a conditioned cost does not itself discharge an outstanding claim or collect cash; see the changing-cost accounting in [Section 13.3](#). An implementation that pays and removes claims earlier must record the payment and liability removal together. It cannot reuse the unmodified holdings identity across a coordinate change.

A.4 Mistakes that change the method

These two panels separate certificate errors from errors in the model, accounts, or execution assumptions. Each points to the relevant test or construction.

Candidate and certificate errors

Shortcut	Consequence and required check	Reference
Treating arbitrary fractional LP solutions as valid outcome columns	A relaxation can admit incoherent means. Keep candidates as mixtures of valid payoffs. An LP lower bound can still certify the same objective; an ideal formulation can describe the correct hull.	Payoff-model tests, Chapter 8 and Section 20.2
Using a merely feasible incumbent as a lower bound	$m_{\text{inc}} \geq m^*$, so $c^\top \mu - m_{\text{inc}} \leq g$. A valid minimization lower bound gives the required upper gap. An exactly minimizing incumbent needs its own optimality justification.	Bound-direction test, Chapter 11
Using the contracted gap in the profit identity	Minimizing over a smaller domain changes the payoff minimum. Certify over the current uncontracted set. A positive original gap can still be safe when $F_\theta(\mu) - g \geq 0$.	Contraction tests, Chapter 12
Returning any coherent candidate	Its associated trade can lose in a valid outcome. Evaluate that trade's direct worst profit or a valid lower certificate before committing it.	Unsafe-reset test, Section 20.1
Using a shifted optimizer objective as economic profit	Restore $C(\theta)$ or use the equivalent direct certificate $m_{\text{lb}} - [C(\hat{\theta}) - C(\theta)]$.	Profit-identity test, Chapter 5 and Section 20.1
Clipping unresolved probabilities to 0/1	Reject unsupported endpoint inverses. Logical settlement changes the compatible set and cost; a reporting floor does not settle a claim.	Endpoint and settlement tests, Chapter 12 and Section 20.4

Model, accounts, and execution errors

Shortcut	Consequence and required check	Reference
Treating a failed witness search as logical impossibility	A timeout leaves feasibility unknown. Logical settlement requires a valid infeasibility or bound proof for the opposite outcome.	Witness-search tests, Section 20.4
Paying out the internal pricing state as trader inventory	This turns virtual offsets into outside liabilities. Pay real holdings and verify that internal corrections leave holdings and cash unchanged.	Ledger tests, Chapter 14 and Section 20.1
Reusing a stale-state certificate	A different state, cost, objective, or outcome version changes the proved trade. Validate the exact snapshot again at commit.	Frozen-state commits, Chapter 19
Treating mixture weights as a unique joint posterior	Several distributions can yield the same traded marginals. The weights certify one feasible representation, not unique posterior inference.	Nonunique mixtures, Section 9.9
Calling a projected last-trade discrepancy executable arbitrage	Check executable bids/asks, available quantities, costs, and permitted fill paths. A price projection contains none of these execution guarantees.	Venue experiments, Chapter 21 ; order-book model, Section 22.6

The implementation milestones are in [Section 21.7](#). They preserve the triangle regression and add tournament fixtures before comparing IND, LCMM, LCMM_LOGICAL, and FWMM. The treatment definitions in [Section 21.1.1](#) make observed settlement common to all four and distinguish it from optional logical implication searches.

Appendix B

Related papers and a reading plan

Use the core readings to derive the market’s cost, feasible price set, correction objective, and oracle certificate. Then choose a follow-up according to the implementation question you want to test.

Status. The bibliography preserves the guide’s 12 October 2026 source checks. Targeted checks on 14 October 2026 covered the recent-work passages, the forecast-error abstract, and the two repository READMEs identified below. These checks did not audit every proof, install external research code, or reproduce its experiments.

B.1 Orientation

The primary paper is Kroer, Dudík, Lahaie, and Balakrishnan’s *Arbitrage-Free Combinatorial Market Making via Integer Programming* (EC 2016); page references below use its 21-page arXiv v2 manuscript of 10 June 2016 [P1]. It combines independent LMSR costs, valid-payoff integer optimization, and fully corrective Frank–Wolfe.

The correction objective and current original-domain gap are defined in the formula sheet (Appendix A); the associated conjugate-gradient trade has the lower profit certificate derived in Section 10.2. This construction differs from a joint-outcome LMSR. The readings below separate costs and prices, coherent expectations and loss, counting and optimization, and boundary and settlement handling.

B.2 The core path

The six-security fixture can be run before completing these readings. Read selected material while deriving its components, and complete these outputs before claiming the guarantees of a larger engine. Each card gives a stopping point rather than requiring an entire paper to be read at once.

1. Charges, prices, and liquidity. Read *Hanson’s logarithmic market scoring rules* [P2] alongside Chapter 2. Produce the cost difference for a finite signed trade and its marginal-price derivative. Calculate the same purchase at two liquidity values and the corresponding initial loss budgets. Check why current quote times quantity differs from the finite charge. Liquidity controls model price response and subsidy; participation and observed market depth require separate evidence. Block modularity becomes relevant when composing these costs.

2. Conjugacy, coherent prices, and fixed-cost loss. Read *Abernethy, Chen, and Vaughan's convex market-making framework* [P3], with [Chapters 4](#) and [7](#). Produce the change of variables from trade to final pricing state, identify $R = C^*$ and its finite domain, and connect the payoff hull to coherent expectations. Derive the fixed-cost loss bound under its initial-account assumptions. Then use the primary FWMM paper for the correction and stopping construction; the general framework does not establish safety of an arbitrary internal reset. The bibliography distinguishes EC 2011 from the expanded 2013 journal treatment.

3. Counting and linear optimization. Read *Chen et al. on combinatorial-market complexity* [P7], followed by the primary paper's payoff representation. For one security language, write both the joint-LMSR partition sum over complete outcomes and $\min_{z \in \mathcal{Z}} c^\top z$. State whether the input lists the outcomes or describes them compactly. The oracle asks a different question from weighted counting; neither this distinction nor an IP encoding gives a general polynomial-time guarantee.

4. The inexpensive baseline's economics. Read *Dudík, Lahaie, and Pennock's LCMM construction* [P4], with [Chapter 14](#) and [Section 21.1.1](#). Produce both valid price inequalities $A\mu \geq a$ and the state-update rule using them. For $\delta = A^\top \eta$, $\eta \geq 0$, check the lower hypothetical-profit bound $a^\top \eta - [C(\theta + \delta) - C(\theta)]$ in one fixed settlement problem. An outer-polytope membership test alone is not an update mechanism. Record the inequality orientation and the numerical acceptance policy; the pilot's capped coordinate solve is a declared approximation.

5. A missing column and an upper gap. Read *Jaggi's Frank–Wolfe analysis* [P5] and the primary paper's master/oracle construction, with [Chapters 9](#) and [11](#). Produce a feasible mixture and derive

$$F_\theta(\mu) - \min_{v \in \mathcal{M}} F_\theta(v) \leq g(\mu)$$

from convexity. Show why a feasible oracle objective and a global minimization lower bound give opposite-sided gap information. Only then study a convergence result: record its algorithm variant, curvature/domain assumptions, and whether it bounds objective error, a selected gap, or a last iterate. The entropy problem has no automatic smooth full-domain rate.

6. Contraction and its certificate domain. Read *Barrier Frank–Wolfe* [P6] and the primary paper's initialization/boundary sections, with [Chapters 12](#) and [13](#). Produce a witness-based coordinate-interior anchor and the mixture $(1 - \varepsilon)v + \varepsilon u$. Distinguish its contracted optimization set from the current uncontracted payoff hull used by the economic gap. Construct a case with zero contracted gap and positive original-domain gap. Compare the source contraction update with the guarded-gap/floor modifications in the triangle programs ([Appendix C](#)); their convergence does not follow merely from citing the source theorem.

B.2.1 A route through the pinned primary paper

The six steps below retain the arXiv v2 page mapping. Personal notation is useful while learning; contributed notes should first give a concordance with $C, R, \mathcal{Z}, \mathcal{M}, F_\theta, g$.

1. §§2.1–2.2, pp. 3–5: derive cost, conjugate, payoff hull, and the correction objective; map the source’s mixed Bregman divergence to $F_\theta(\mu)$ (Chapters 2 to 5).
2. §2.3 and §4.1, pp. 5, 10–12: connect the integer oracle to the restricted master (Chapters 8 and 9).
3. §4.2 and Appendix D, pp. 12–13, 21: derive the correction-profit identity and check it on the triangle (Section 10.2 and Chapter 15).
4. §3.2, §4.3, and Appendices B–C: identify the settlement cost change, entropy endpoints, and anchor requirements (Chapters 7, 12 and 13).
5. §3.1, pp. 6–9: implement the tournament payoff language. For $X_1 = 1, X_2 = 2$, compute $Y = X_2 - X_1 = 1$ and verify that $X_2 > X_1$ uses the positive sign. Chapter 8 records the apparent reversed comparison-price initialization in the printed source; preserve that source qualification rather than silently changing a quoted formula.
6. §§5–6, pp. 14–18: read the empirical findings with their replay assumptions and runtime qualifications (Chapter 17).

B.3 Choose a follow-up by its implementation question

For an expensive oracle, begin with active-set scheduling; for a restricted security language, first check whether a specialized market design supports exactly the required operations. The grouped table gives one output for each reading. These are proposed learning and experimental tasks, not claims that the cited implementation has been reproduced here.

Table B.1: Follow-up readings grouped by the decision they inform.

Reading and question	Output and transfer check
Improve optimization	
<i>Frank–Wolfe variants</i> (2015) [P10]: which step is faster?	Compare vanilla, pairwise, and fully corrective steps at equal certificate targets. Map objective regularity, polytope geometry, oracle accuracy, master accuracy, and convergence quantity before transferring a rate. Redundant weight representations require their own check; speed alone is empirical.
<i>Blended Conditional Gradients</i> (2019) [P12]: when can active-set work replace oracle calls?	Measure total elapsed time, oracle/master costs, and achieved certificates, including unchanged returns. Every market acceptance still needs a currently valid original-domain lower bound; a package success flag does not supply it.
Change the security language or inference structure	
<i>U.S. election market</i> (2013) [P8]: which constraints propagate information?	Compare atomic, sum, and comparison securities and their propagation constraints with the tournament language. Measure changed prices after one common order; do not infer a later FWMM deployment from this application.

Reading and question	Output and transfer check
<i>Interval securities</i> (2021) [P13]: can a specialized representation answer the needed operation?	Implement the same interval family in both representations; report query or trade time, storage, and preprocessing. Check the supported operation and family before claiming a comparison with generic IP.
<i>Geometric viewpoint</i> (2025; 2024 preprint) [P14]: which set structure makes pricing tractable?	Identify the supported set family and input representation, then compare the same operation including preprocessing. Record securities, explicit outcomes, compact-model size, and query order separately.
<i>Graphical-model market maker</i> (2018) [P16]: what does graph structure permit?	Identify graph restrictions and exact versus approximate operations before timing a common small model. Retain the source’s mechanism and loss assumptions instead of replacing them with the FWMM certificate.
Evaluate information aggregation	
<i>Forecast-error decomposition</i> (2017; 2018 revision) [P11]: which error component changes?	Extract the exponential-utility and exponential-family belief assumptions identified in the checked abstract. Map changes in signals, liquidity, and horizon to sampling, maker-bias, and convergence components. The pilot’s fresh budget/limit traders differ; wider transfer needs the full assumptions.
Model settlement or venue execution	
<i>Decreasing utility for information</i> (2014) [P9]: which cost transformation preserves the accounts?	Compare the source construction with C_σ in Section 13.2, identify its domain restriction, and verify one cash/liability identity across settlement. Arbitrary conditional-price replacement does not follow from this background reference.
<i>Probabilistic forest</i> (2025) [P15]: which apparent opportunity survives executable prices?	Reproduce the chosen bid/ask, depth, fee, and fill assumptions in a separate venue study. Report the worst-payoff certificate or its absence; this is not a replay through a cost-function engine (Section 21.6).

For complexity comparisons, name the number of securities, explicit outcome count N , compact representation size n , query/parlay order, and preprocessing cost. As an elementary illustration, $N^{1/2} = 2^{n/2}$ when $N = 2^n$. This is not a claimed bound for either specialized-market paper; it explains why sublinear explicit-size time need not be polynomial in a compact input. Specialized exact pricing remains useful within its stated family.

B.4 Recent adjacent work and the scope of its guarantees

ParlayMarket: model, computation, and loss. The targeted content check used [arXiv v3 of 10 August 2026](#); the AFT 2026 proceedings record is separately dated 2 October 2026 [P17]. Its pairwise Ising/exponential-family joint model induces coherent expectations under exact inference. The historical replay uses loopy belief propagation. That numerical procedure and the exact model are different objects.

The checked loss discussion concerns expected aggregate loss in a stochastic model where informed traders target Gaussian-score conditional probabilities and noise traders use noisy score observations. Its complete-parlay scaling result uses uniform flow; it also studies learning toward a pairwise target. Those claims do not establish the feasible-mixture and pathwise correction-profit certificate used here. A controlled comparison should hold payoffs and informed-signal draws fixed, report inference residuals and expected loss separately, and explicitly record where the trading rules differ. These are targeted model, replay, and guarantee-scope checks, not a proof audit.

LOPMM: operation size and restricted flow. The checked [v3 preprint of 5 October 2026](#) uses the name LOPMM; earlier versions used APMM [P18]. For an m -leg parlay, its stated arithmetic costs distinguish quoting, $\Theta(4^m)$, from a full trade, $\Theta(5^m)$, under its hierarchical representation. These are operation counts, not observed latency guarantees.

Its worst-case $O(M^2)$ loss statement for M base events assumes low-order flow, bounded log-probability displacement, and partially informed honest reports. The fixed-truth expected $O(M)$ result additionally uses local dependence; that expected result does not require the low-order assumption. Compare those conditions explicitly with arbitrary signed flow before transferring a guarantee. A first comparison can vary parlay order while holding base-event count and truthful information fixed, then separately violate the flow assumptions. No peer-reviewed venue was verified for this preprint.

B.5 Available implementation references

The [clinicalml/fw-inference](#) README describes Barrier Frank–Wolfe marginal inference, experiment configuration, and MAP-oracle interfaces [S3]. Its checked requirements name Python 2.7, NumPy/SciPy, and Gurobi, with optional inference libraries. Use it to study contraction and inference; the proposed order, cash/liability, settlement, and transactional-commit contracts still need implementation and validation.

The [pokutta/bcg](#) README exposes `bcg.run_BCG.BCG`, objective/gradient inputs, LP or custom feasible regions, and performance logging [S4]. It is a reference for active-set and oracle scheduling. Its returned dual information must be mapped to the current market’s coefficient vector, scale, domain, and snapshot before use. Neither external repository was installed or run for this revision; the 14 October checks covered landing/README material, not a source-code audit.

The local program in [Appendix C](#) is a six-security NumPy/SciPy fixture using the documented `mi.lp` result fields [S1]. Its historical snapshot and separately versioned hardened variant have distinct results ([Chapter 16](#)). Neither supplies the proposed tournament simulator or multi-module market engine.

B.6 How to use the bibliography

Tags beginning P identify research papers; S identifies software and documentation; B identifies textbook background; W identifies writing guidance; and E identifies deployment or venue evidence. The core path uses [P1; P2; P3; P4; P5; P6; P7]; later decisions use [P8; P9; P10; P11; P12; P13; P14; P15; P16; P17; P18]. The convex-optimization text [B1] and Bertsekas’s writing guidance [W1] are references to consult as needed. Full

entries and descriptive links preserve the version relevant to each task. Standalone chapters retain their own source lists. When contributing a citation, record the question it supports; link-maintenance and source-conversion history belong in contributor or release metadata.

Discussion questions.

1. Starting from conjugacy and a valid oracle lower bound, derive one lower correction-profit certificate and check its direction on the triangle.
2. Choose one optimization follow-up and compare its domain, regularity, oracle, master accuracy, and stopping conclusion with the actual implementation. Mark every unmet or unchecked assumption before making a convergence claim.
3. For a specialized market design, name the supported operation and all relevant input-size parameters. Give one common experiment and one reason its guarantee cannot yet be transferred to the proposed engine.

Appendix C

Complete listing of the example program

Consult the complete six-security experiment used in [Chapter 16](#), with the same source and line numbers.

The program checks the payoff model, correction objective, gradient, profit certificate, and a fixed-cost three-trade ledger against all eight outcomes. The ledger uses $\min(\alpha, 0.95)$. Its six-coordinate model and audit log are fixture-specific. The all-zero/all-one initialization supplies coordinate-interior means; general witness coverage, logical settlement, LCMM, incomplete MIP solves, concurrent commits, and venue execution remain outside its scope. **Status.** The unchanged [historical source file](#) generates this listing and all historical excerpts in [Chapter 16](#). The guide records Python 3.12.14, NumPy 2.3.5, and SciPy 1.17.0; its original raw JSON was not preserved. Source hashes and later executions are in [the revision manifest](#). The review packet’s reconstructed listing has different bytes; the [source comparison](#) records its relationship to the repository artifact.

From the tutorial directory, run the plain source without `-O`:

```
| python code/fwmm_triangle.py > results.json
```

Installation, solver provenance, and function contracts are in [Sections 16.2](#) and [16.3](#). PDF line numbers and continuation arrows are printing aids; execute the linked source.

Historical API	Mathematical meaning
oracle	Integer linear minimization oracle (LMO) over valid payoffs.
entropy	Convex conjugate on the binary box: negative entropy.
divergence	Correction objective, including the original cost constant.
state	Finite inverse pricing state for strict binary-interior prices.

Caution. The historical gap clamp can turn a NaN bound into zero. Assignment checks omit explicit box bounds, Boolean membership conflates infeasibility with solver failure, zero iterations are unsupported, and `accuracy threshold` merges two stop conditions. Its ordinary assertions are disabled by `-O`. Use the separate [versioned hardened implementation](#) and its [new execution record](#) for the checked interfaces and their limits ([Section 16.13](#)). The guarded contraction update remains an engineering modification, without an unchanged convergence theorem ([Chapter 12](#)). Neither version implements a production engine.

```

1  #!/usr/bin/env python3
2  """Pedagogical FWMM experiment; NumPy/SciPy only, not an NCAA replication.
3
4  Usage: python fwmm_triangle.py [--b 1.0] [--alpha 0.9999]
5  Prints JSON containing numerical checks, the FW trace, and the final
6  ↪ certificate.
7  """
8  import argparse
9  import itertools
10 import json
11 import platform
12
13 import numpy as np
14 import scipy
15 from scipy.optimize import Bounds, LinearConstraint, linprog, milp, minimize
16 from scipy.special import expit, xlogy
17
18 PAIRS = [(0, 1), (1, 2), (0, 2)]
19 N = 6 # x1, x2, x3, x1*x2, x2*x3, x1*x3; each is a Yes security
20 A, rhs = [], []
21 for k, (i, j) in enumerate(PAIRS, start=3):
22     for parent in (i, j):
23         row = np.zeros(N)
24         row[k], row[parent] = 1, -1
25         A.append(row)
26         rhs.append(0.0)
27     row = np.zeros(N)
28     row[i], row[j], row[k] = 1, 1, -1
29     A.append(row)
30     rhs.append(1.0)
31 A, rhs = np.array(A), np.array(rhs)
32 BOUNDS = Bounds(np.zeros(N), np.ones(N))
33 CONSTRAINTS = LinearConstraint(A, -np.inf, rhs)
34 ORACLE_AUDIT = []
35
36
37 def oracle(c):
38     """Exact-to-solver-tolerances integer LMO; fail closed without
39     ↪ optimality."""
40     scale = max(float(np.max(abs(c))), 1e-100)
41     result = milp(c/scale, integrality=np.ones(N), bounds=BOUNDS,
42                  constraints=CONSTRAINTS,
43                  options={"mip_rel_gap": 0.0, "presolve": True})
44     if result.status != 0 or result.x is None:
45         raise RuntimeError("Oracle did not prove optimality: " +
46                            ↪ result.message)
47     z = np rint(result.x)
48     if (np.max(abs(z - result.x)) > 1e-7 or
49         np.max(A @ z - rhs) > 1e-7):
50         raise RuntimeError("Invalid integer oracle solution")
51     # Floating-point MIP optimality has tolerances. This guard is a pragmatic
52     # numerical allowance, not an exact-arithmetic theorem. Every returned
53     # bound is subsequently audited by enumeration in this tiny experiment.
54     lower_bound = scale * (float(result.mip_dual_bound)-2e-6)
55     incumbent = float(c @ z)
56     ORACLE_AUDIT.append((c.copy(), lower_bound))
57     return z, lower_bound, incumbent

```

```

56
57
58 def cost(theta, b):
59     return float(b * np.logaddexp(0.0, theta / b).sum())
60
61
62 def entropy(mu, b):
63     # xlogy defines 0*log(0)=0, so entropy is continuous at endpoints.
64     return float(b * (xlogy(mu, mu) + xlogy(1-mu, 1-mu)).sum())
65
66
67 def state(mu, b):
68     # Never clip: endpoint prices require a different, settled cost function.
69     if np.any(mu <= 0) or np.any(mu >= 1):
70         raise ValueError("Finite state requires every marginal strictly in
71             ↪ (0,1)")
72     return b * (np.log(mu) - np.log1p(-mu))
73
74
75 def divergence(mu, theta, b):
76     return entropy(mu, b) + cost(theta, b) - float(theta @ mu)
77
78
79 def correct(active, u, eps, theta, b, initial_weights=None):
80     """Minimize over the convex hull of contracted active vertices."""
81     contracted = (1-eps) * np.asarray(active) + eps * u
82     k = len(contracted)
83     w0 = np.full(k, 1/k) if initial_weights is None else initial_weights
84     objective = lambda w: divergence(w @ contracted, theta, b)
85     gradient = lambda w: contracted @ (state(w @ contracted, b) - theta)
86     result = minimize(objective, w0, jac=gradient, method="SLSQP",
87         bounds=Bounds(np.zeros(k), np.ones(k)),
88         constraints=LinearConstraint(np.ones((1, k)), 1, 1),
89         options={"ftol": 1e-13, "maxiter": 2000})
90     if not result.success:
91         raise RuntimeError("Active-set solve failed: " + result.message)
92     # Repair only solver roundoff, retaining an explicitly feasible mixture.
93     if min(result.x) < -1e-8 or abs(sum(result.x)-1) > 1e-8:
94         raise RuntimeError("Invalid active-set weights")
95     w = np.maximum(result.x, 0)
96     w /= w.sum()
97     return w @ contracted, w
98
99
100 def project_fw(theta, b, alpha, max_iterations=200):
101     # For this specific model, these two valid vertices cover 0 and 1 in
102     # every coordinate; their mean suffices for the paper's coordinate
103     # interior condition. A general model needs the full InitFW routine.
104     active = [oracle(np.ones(N))[0], oracle(-np.ones(N))[0]]
105     u = np.mean(active, axis=0)
106     eps, weights, best, trace = 0.1, None, None, []
107     stop_reason = "iteration budget"
108     for iteration in range(1, max_iterations+1):
109         mu, weights = correct(active, u, eps, theta, b, weights)
110         new_theta = state(mu, b)
111         c = new_theta-theta
112         z, lower_bound, incumbent = oracle(c)
113         f = divergence(mu, theta, b)

```

```

113     # A minimization lower bound gives an UPPER bound on the FW gap.
114     g_upper = max(0.0, float(c @ mu)-lower_bound)
115     g_incumbent = float(c @ mu)-incumbent
116     profit_bound = f-g_upper
117     record = dict(iteration=iteration, active_vertices=len(active),
118                  contraction=eps, divergence=f, gap_upper=g_upper,
119                  gap_incumbent=g_incumbent,
120                  profit_lower_bound=profit_bound)
121     trace.append(record)
122     if best is None or profit_bound > best["profit_lower_bound"]:
123         best = dict(record, prices=mu.copy(), theta=new_theta.copy(),
124                    weights=weights.copy(),
125                    active=np.asarray(active).copy())
126     if g_upper <= (1-alpha)*f or f <= 1e-12:
127         stop_reason = "accuracy threshold"
128         break
129     if not any(np.array_equal(z, old) for old in active):
130         active.append(z)
131         weights = np.append(weights, 0.0)
132         gu = float(c @ (mu-u))
133         # Using the guarded upper gap here is an engineering modification,
134         # not a verbatim Algorithm 2 implementation or a convergence proof.
135         if gu < 0 and g_upper / (-4*gu) < eps:
136             eps = min(g_upper / (-4*gu), eps/2)
137     if best["profit_lower_bound"] < 0:
138         return theta.copy(), best, trace, "unchanged: " + stop_reason
139     return best["theta"], best, trace, "safe update: " + stop_reason
140
141 def membership(mu, vertices):
142     result = linprog(np.zeros(len(vertices)),
143                    A_eq=np.vstack([np.ones(len(vertices)), vertices.T]),
144                    b_eq=np.r_[1, mu], bounds=(0, None), method="highs")
145     return bool(result.success)
146
147 def ledger_test(vertices, b, alpha):
148     """Actual trader liabilities q and cash are separate from internal
149     ↪ theta."""
150     theta, q = np.zeros(N), np.zeros(N)
151     cash, total_internal_profit = 0.0, np.zeros(len(vertices))
152     loss_bound = N*b*np.log(2) # initial theta=0; R(z)=0 for every outcome
153     trades = [b*np.r_[np.zeros(3), np.full(3, np.log(.05/.95))],
154              b*np.array([.2, -.1, .05, -.3, -.2, -.4]),
155              b*np.array([-0.05, .1, -.15, -.2, -.4, -.2])]
156     records = []
157     for step, delta in enumerate(trades, 1):
158         payment = cost(theta+delta, b)-cost(theta, b)
159         cash += payment
160         q += delta
161         theta += delta
162         loss = vertices @ q-cash
163         assert max(loss) <= loss_bound+1e-9*b
164         old_theta = theta.copy()
165         theta, _, _, action = project_fw(old_theta, b, alpha)
166         internal_profit = (vertices @ (theta-old_theta)
167                          - cost(theta, b)+cost(old_theta, b))
168         assert min(internal_profit) >= -1e-9*b

```

```

168     total_internal_profit += internal_profit
169     # Internal corrections change neither real trader holdings nor cash.
170     hypothetical_base_loss = (vertices @ theta-cost(theta, b)
171                             + cost(np.zeros(N), b))
172     assert
173     ↪ np.max(abs(loss-(hypothetical_base_loss-total_internal_profit))) <
174     ↪ 1e-9*b
175     assert max(loss) <= loss_bound-min(total_internal_profit)+1e-9*b
176     records.append(dict(trade=step, customer_payment=payment,
177                        actual_worst_loss=float(max(loss)), action=action,
178                        internal_worst_profit=float(min(internal_profit))))
179     return dict(alpha=alpha, original_loss_bound=loss_bound, actual_cash=cash,
180                actual_trader_positions=q.tolist(),
181                ↪ internal_state=theta.tolist(),
182                ↪ final_losses_by_outcome=loss.tolist(),
183
184     ↪ cumulative_internal_profit_min=float(min(total_internal_profit)),
185
186     ↪ tightened_loss_bound=loss_bound-float(min(total_internal_profit)),
187     ↪ trades=records)
188
189 def main():
190     parser = argparse.ArgumentParser(description=__doc__)
191     parser.add_argument("--b", type=float, default=1.0)
192     parser.add_argument("--alpha", type=float, default=0.9999)
193     args = parser.parse_args()
194     b, alpha = args.b, args.alpha
195     if b <= 0 or not 0 < alpha < 1:
196         parser.error("require b > 0 and 0 < alpha < 1")
197     vertices = np.array([list(x) + [x[i]*x[j] for i, j in PAIRS]
198                        for x in itertools.product((0.0, 1.0), repeat=3)])
199     binary_vectors = np.array(list(itertools.product((0.0, 1.0), repeat=N)))
200     admitted = [z for z in binary_vectors if np.all(A @ z <= rhs+1e-12)]
201     assert {tuple(z) for z in admitted} == {tuple(z) for z in vertices}
202     p0 = np.array([0.5]*3 + [0.05]*3)
203     theta = state(p0, b)
204     # LP relaxation accepts p0, but the exact convex hull does not.
205     assert np.max(A @ p0-rhs) <= 1e-12
206     assert not membership(p0, vertices)
207     separator = np.array([-1.0]*3 + [1.0]*3)
208     lp = linprog(separator, A_ub=A, b_ub=rhs, bounds=(0, 1), method="highs")
209     _, ip_bound, ip_value = oracle(separator)
210     assert abs(ip_value - min(vertices @ separator)) < 1e-10
211     assert abs(lp.fun+1.5) < 1e-10 and abs(ip_value+1) < 1e-10
212
213     # Enumeration reference: all eight vertices, numerical contraction 1e-12.
214     reference, _ = correct(vertices, vertices.mean(axis=0), 1e-12, theta, b)
215     # Symmetry and the active inequality 3a-3c=1 also give a closed form.
216     a = (1+np.sqrt(7))/9
217     analytic = np.array([a]*3 + [a-1/3]*3)
218     assert np.max(abs(reference-analytic)) < 2e-6
219     reference_value = divergence(analytic, theta, b)
220
221     # Central differences check the derivative used by the LMO.
222     test_mu = np.array([0.3, 0.6, 0.4, 0.2, 0.2, 0.2])
223     step, eye = 1e-6, np.eye(N)
224     fd = np.array([(divergence(test_mu+step*e, theta, b)

```

```

221         - divergence(test_mu-step*e, theta, b))/(2*step) for e in
    ↪ eye])
222 gradient_error = float(np.max(abs(fd-(state(test_mu, b)-theta))))
223 assert gradient_error < 1e-6*b
224
225 final_theta, best, trace, action = project_fw(theta, b, alpha)
226 final_prices = expit(final_theta/b)
227 delta = final_theta-theta
228 profits = vertices @ delta - (cost(final_theta, b)-cost(theta, b))
229 f = divergence(final_prices, theta, b)
230 g_exact = float(delta @ final_prices - min(vertices @ delta))
231 assert membership(final_prices, vertices)
232 assert abs(min(profits)-(f-g_exact)) < 1e-10*b
233 assert min(profits) >= -1e-10*b
234 if action == "safe update: accuracy threshold":
235     assert min(profits) >= alpha*reference_value-1e-9*b
236     # Bernoulli negative entropy is 4b-strongly convex in Euclidean norm.
237     assert np.linalg.norm(final_prices-analytic) <=
    ↪ np.sqrt(best["gap_upper"]/(2*b))+1e-6
238
239     # A coherent target alone is insufficient to preserve the loss guarantee.
240     unsafe_prices = vertices.mean(axis=0)
241     unsafe_theta = state(unsafe_prices, b)
242     unsafe_delta = unsafe_theta-theta
243     unsafe_profit = float(min(vertices @ unsafe_delta)
    ↪ - cost(unsafe_theta, b)+cost(theta, b))
244     assert membership(unsafe_prices, vertices) and unsafe_profit < 0
245     ledger = ledger_test(vertices, b, min(alpha, .95))
246     early_theta, _, _, early_action = project_fw(theta, b, alpha,
    ↪ max_iterations=1)
247     late_theta, _, _, late_action = project_fw(theta, b, alpha,
    ↪ max_iterations=7)
248     assert np.array_equal(early_theta, theta)
249     late_profit = float(min(vertices @ (late_theta-theta)
    ↪ - cost(late_theta, b)+cost(theta, b))
250     assert late_profit > 0 and membership(expit(late_theta/b), vertices)
251     bound_audit_error = max(lb-float(min(vertices @ c)) for c, lb in
    ↪ ORACLE_AUDIT)
252     assert bound_audit_error <= 1e-10*b
253     output = dict(
254         environment=dict(python=platform.python_version(),
    ↪ numpy=np.__version__,
255         scipy=scipy.__version__), b=b, alpha=alpha,
256         security_order=["x1", "x2", "x3", "y12", "y23", "y13"],
257         ip_structural_vectors_checked=len(binary_vectors),
258         ip_structural_valid_vectors=len(admitted),
259         initial_prices=p0.tolist(), initial_local_constraint_max_violation=0.0,
260         initial_exact_hull_membership=False,
261         triangle_inequality_violation=float(sum(p0[:3])-sum(p0[3:])-1),
262         lp_oracle_value=float(lp.fun), ip_oracle_value=ip_value,
263         ip_oracle_lower_bound=ip_bound, lp_oracle_solution=lp.x.tolist(),
264         enumerated_reference_prices=reference.tolist(),
265         analytic_reference_prices=analytic.tolist(),
266         optimal_arbitrage_profit=reference_value,
267         ↪ gradient_max_error=gradient_error,
268         fw_action=action, fw_final_prices=final_prices.tolist(),
269         fw_best_iteration=best["iteration"], fw_iterations=len(trace),
270         fw_divergence=f, fw_exact_gap=g_exact,

```

```
272     fw_profit_lower_bound=best["profit_lower_bound"],
273     fw_worst_outcome_profit=float(min(profits)),
274     fw_all_outcome_profits=profits.tolist(),
275     coherent_unsafe_target=unsafe_prices.tolist(),
276     coherent_unsafe_update_worst_profit=unsafe_profit,
277     oracle_calls_audited=len(ORACLE_AUDIT),
278     max_oracle_lower_bound_minus_exact_min=bound_audit_error,
279     ledger=ledger, interruption_tests=dict(early_action=early_action,
280     late_action=late_action, late_worst_outcome_profit=late_profit),
281     fw_trace=trace)
282     print(json.dumps(output, indent=2))
283
284
285 if __name__ == "__main__":
286     main()
```

Listing C.1: The complete program code/`fwmm_triangle.py`

Appendix D

Source concordance and artifacts

Locate the source of an argument and the files needed to check it.

D.1 Paper locations

The primary source is the 21-page arXiv v2 manuscript of Kroer et al. [P1]. Page numbers below count its title page as page 1. The tutorial uses $F_\theta(\mu)$ for the paper’s $D(\mu||\theta)$, $\mathcal{M}$ for its marginal polytope, and $\mathcal{Z}$ for valid payoff vectors. The notation crosswalk gives the remaining correspondences. Statements reproduced from the paper retain their source number. Source restatements are not independent tutorial theorems.

Tutorial	Source location	Qualification or local concordance
Chapters 2 to 4	§2, pp. 3–6	Signed trades, conjugacy, and coherent means
Chapter 6	Prop. 2.4, App. A, pp. 5, 19	Section 6.7; value and attainment have different scopes
Chapter 7	Prop. B.1, pp. 19–20	Fixed-cost loss bound
Chapter 8	§3.1, pp. 6–8	Section 8.7; comparison-sign qualification precedes source wording
Chapter 9	§4.1, Alg. 2, pp. 10–12	Section 9.11; smooth-rate statement also uses [P5]
Chapter 10	Prop. 4.1, App. D, pp. 13, 21	Section 10.6; candidate and stopping certificates
Chapter 11	Tutorial extension	Numerical solver bounds extend the exact-oracle argument
Chapter 12	Alg. 3, App. C, pp. 13, 21	Section 12.5; active-coordinate entropy domain
Chapter 13	Def. 3.1, Eq. (5), Prop. B.2, pp. 9, 20	Section 13.4; compatible restrictions
Chapter 14	Mechanism 1, p. 10	Approximate accepted corrections and outside accounts
Chapters 15 and 16	Tutorial example	Independent analytic benchmark and historical script
Chapter 17	§5–6, pp. 14–17	Historical replay, reconstructed orders, and reported limits

D.2 Source-note lookup

The repository contains the [original Markdown notes and their index](#). M identifiers refer to its `math/` directory, G identifiers to `guide/`, and E identifiers to `evidence/`. These identifiers are internal source-note names; bibliography tags identify cited publications. The repository README provides the full filenames.

Chapter	Source notes	Chapter	Source notes
1	G01	12	G07; M06–M07 (boundaries)
2	M01; G02 (trades)	13	G07; M07 (settlement)
3	M02; G03 (coherence)	14	G06; M07 (ledgers); G03
4	M03	15	G10
5	M04; G03 (objective)	16	G11; Python module
6	M05; G03; G17	17	G09
7	M07; G02 (budget)	18	E01–E05
8	G04	19	G12
9	G05; M06; G03	20	G13
10	G06; M06	21	G14; E06
11	G08; M06	22	G15

D.3 Executable and revision artifacts

The historical program is `code/fwmm_triangle.py`, printed in [Appendix C](#). Its bytes remain unchanged by this revision. [The output provenance note](#) records that the original JSON was not preserved. The reported historical numbers must not be interpreted as outputs of a new execution.

The separate [hardened implementation](#) and its [regression tests](#) validate numerical inputs, solver results, coherent candidates, and returned-state metadata. The manifest records new runs in the available environment separately from the historical observations.

The revision supplies a separate [arithmetic and certificate-policy checker](#), its [recorded results](#), and an [artifact manifest](#). The [issue ledger](#) records implementation or retention decisions for every review finding. These checks cover finite examples and failure cases; they do not prove a floating-point solver bound for an arbitrary MIP or establish convergence of a modified algorithm.

References

- [P1] Christian Kroer, Miroslav Dudík, Sébastien Lahaie, and Sivaraman Balakrishnan. “Arbitrage-Free Combinatorial Market Making via Integer Programming.” In: *Proceedings of the 2016 ACM Conference on Economics and Computation (EC '16)*. This document follows the arXiv v2 manuscript of 10 June 2016; page numbers count its title page as page 1. Author page: <https://www.columbia.edu/~ck2945/publication/kroer-2016-arbitrage/>. 2016. DOI: [10.1145/2940716.2940767](https://doi.org/10.1145/2940716.2940767). arXiv: [1606.02825v2](https://arxiv.org/abs/1606.02825v2) [cs.GT] (cit. on pp. i, 2, 3, 5–8, 11, 12, 15, 16, 19, 23, 27–29, 33, 34, 36, 37, 42, 44–46, 50–52, 59–61, 64–66, 68, 70, 95–99, 103, 109, 122, 134, 135, 142, 146, 155).
- [P2] Robin Hanson. “Logarithmic Market Scoring Rules for Modular Combinatorial Information Aggregation.” In: *Journal of Prediction Markets* 1.1 (2007). The linked author manuscript is dated January 2002; 2007 is the journal year cited by the paper, pp. 3–15. URL: <https://hanson.gmu.edu/mktscore.pdf> (visited on Oct. 12, 2026) (cit. on pp. 7, 19, 142, 146).
- [P3] Jacob Abernethy, Yiling Chen, and Jennifer Wortman Vaughan. “An Optimization-Based Framework for Automated Market-Making.” In: *Proceedings of the 12th ACM Conference on Electronic Commerce (EC '11)*. Expanded journal treatment: *Efficient Market Making via Convex Optimization, and a Connection to Online Learning*, ACM Transactions on Economics and Computation 1(2), Article 12, 2013, <https://doi.org/10.1145/2465769.2465777>. 2011. arXiv: [1011.1941](https://arxiv.org/abs/1011.1941) (cit. on pp. 6, 11, 12, 143, 146).
- [P4] Miroslav Dudík, Sébastien Lahaie, and David M. Pennock. “A Tractable Combinatorial Market Maker Using Constraint Generation.” In: *Proceedings of the 13th ACM Conference on Electronic Commerce (EC '12)*. 2012, pp. 459–476. DOI: [10.1145/2229012.2229047](https://doi.org/10.1145/2229012.2229047). URL: <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/DudikLaPe12.pdf> (visited on Oct. 12, 2026) (cit. on pp. 39, 68, 143, 146).
- [P5] Martin Jaggi. “Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization.” In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. Vol. 28. PMLR 1. 2013, pp. 427–435. URL: <https://proceedings.mlr.press/v28/jaggi13.html> (visited on Oct. 12, 2026) (cit. on pp. 42, 45, 46, 143, 146, 155).
- [P6] Rahul G. Krishnan, Simon Lacoste-Julien, and David Sontag. “Barrier Frank-Wolfe for Marginal Inference.” In: *Advances in Neural Information Processing Systems 28 (NIPS)*. 2015. arXiv: [1511.02124](https://arxiv.org/abs/1511.02124) (cit. on pp. 59, 143, 146).

- [P7] Yiling Chen, Lance Fortnow, Nicolas Lambert, David M. Pennock, and Jennifer Wortman. “Complexity of Combinatorial Market Makers.” In: *Proceedings of the 9th ACM Conference on Electronic Commerce (EC ’08)*. 2008. arXiv: [0802.1362](#) (cit. on pp. 45, 143, 146).
- [P8] Miroslav Dudík, Sébastien Lahaie, David M. Pennock, and David Rothschild. “A Combinatorial Prediction Market for the U.S. Elections.” In: *Proceedings of the 14th ACM Conference on Electronic Commerce (EC ’13)*. 2013, pp. 341–358. DOI: [10.1145/2492002.2482601](#). URL: <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ec62-dudik.pdf> (visited on Oct. 12, 2026) (cit. on pp. 39, 144, 146).
- [P9] Miroslav Dudík, Rafael Frongillo, and Jennifer Wortman Vaughan. “Market Making with Decreasing Utility for Information.” In: *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI)*. 2014. arXiv: [1407.8161](#) (cit. on pp. 65, 145, 146).
- [P10] Simon Lacoste-Julien and Martin Jaggi. “On the Global Linear Convergence of Frank-Wolfe Optimization Variants.” In: *Advances in Neural Information Processing Systems 28 (NIPS)*. 2015. arXiv: [1511.05932](#) (cit. on pp. 135, 144, 146).
- [P11] Miroslav Dudík, Sébastien Lahaie, Ryan Rogers, and Jennifer Wortman Vaughan. “A Decomposition of Forecast Error in Prediction Markets.” In: *Advances in Neural Information Processing Systems 30 (NIPS)*. The 2018 arXiv revision does not change the conference year. 2017. arXiv: [1702.07810](#) (cit. on pp. 129, 136, 145, 146).
- [P12] Gábor Braun, Sebastian Pokutta, Dan Tu, and Stephen Wright. “Blended Conditional Gradients.” In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Vol. 97. PMLR. 2019, pp. 735–743. URL: <https://proceedings.mlr.press/v97/braun19a.html> (visited on Oct. 12, 2026) (cit. on pp. 135, 144, 146).
- [P13] Miroslav Dudík, Xintong Wang, David M. Pennock, and David M. Rothschild. “Log-time Prediction Markets for Interval Securities.” In: *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2021. arXiv: [2102.07308](#) (cit. on pp. 135, 145, 146).
- [P14] Prommy Sultana Hossain, Xintong Wang, and Fang-Yi Yu. “Designing Automated Market Makers for Combinatorial Securities: A Geometric Viewpoint.” In: *Proceedings of the 2025 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Preprint first posted in 2024. 2025, pp. 1329–1365. DOI: [10.1137/1.9781611978322.40](#). arXiv: [2411.08972](#) (cit. on pp. 135, 145, 146).
- [P15] Oriol Saguillo, Vahid Ghafouri, Lucianna Kiffer, and Guillermo Suarez-Tangil. “Unravelling the Probabilistic Forest: Arbitrage in Prediction Markets.” In: *7th Conference on Advances in Financial Technologies (AFT 2025)*. Vol. 354. LIPIcs. Cited as source [16] in the evidence notes. 2025, 27. DOI: [10.4230/LIPIcs.AFT.2025.27](#). arXiv: [2508.03474](#) (cit. on pp. 106, 136, 145, 146).
- [P16] Kathryn Blackmond Laskey, Wei Sun, Robin Hanson, Charles Twardy, Shou Matsumoto, and Brandon Goldfedder. “Graphical Model Market Maker for Combinatorial Prediction Markets.” In: *Journal of Artificial Intelligence Research* 63 (2018), pp. 421–460. DOI: [10.1613/jair.1.11249](#) (cit. on pp. 135, 145, 146).

- [P17] Ranvir Rana, Viraj Nadkarni, Niusha Moshrefi, and Pramod Viswanath. “ParlayMarket: Automated Market Making for Parlay-Style Joint Contracts.” In: *8th Conference on Advances in Financial Technologies (AFT 2026)*. Vol. 395. LIPIcs. Proceedings published 2 October 2026. 2026, 15. doi: [10.4230/LIPIcs.AFT.2026.15](https://doi.org/10.4230/LIPIcs.AFT.2026.15). arXiv: [2603.22596v3](https://arxiv.org/abs/2603.22596v3) (cit. on pp. 145, 146).
- [P18] Niusha Moshrefi, Ranvir Rana, and Pramod Viswanath. *LOPMM: A Low-Order Parlay Market Maker*. Preprint, version 3 of 5 October 2026; no peer-reviewed venue verified. 2026. arXiv: [2607.18299v3](https://arxiv.org/abs/2607.18299v3). URL: <https://arxiv.org/html/2607.18299v3> (visited on Oct. 12, 2026) (cit. on p. 146).
- [S1] *scipy.optimize.milp reference documentation*. Version 1.17.0, matching the tested package. Relevant fields: `integrality`, `time_limit`, `mip_rel_gap`, solver status, `mip_dual_bound`. SciPy. URL: <https://docs.scipy.org/doc/scipy-1.17.0/reference/generated/scipy.optimize.milp.html> (visited on Oct. 12, 2026) (cit. on pp. 54, 55, 78, 80, 146).
- [S2] *Model attributes and attribute types*. Relevant attributes: `ObjVal`, `ObjBound`, `MIPGap`. Attribute types: <https://docs.gurobi.com/projects/optimizer/en/current/concepts/attributes/types.html>. Gurobi Optimization. URL: <https://docs.gurobi.com/projects/optimizer/en/current/reference/attributes/model.html> (visited on Oct. 12, 2026) (cit. on pp. 54, 55).
- [S3] Rahul G. Krishnan et al. *Barrier Frank-Wolfe inference code*. Author-associated repository; identity and stated legacy requirements checked, not installed or reproduced. URL: <https://github.com/clinicalml/fw-inference> (visited on Oct. 12, 2026) (cit. on pp. 98, 146).
- [S4] Sebastian Pokutta et al. *Blended conditional-gradient code*. An optimization reference, not a full combinatorial market engine. URL: <https://github.com/pokutta/bcg> (visited on Oct. 12, 2026) (cit. on p. 146).
- [B1] Stephen Boyd and Lieven Vandenbergh. *Convex Optimization*. Section 3.3: conjugate functions, supporting-line geometry, log-sum-exp example. Cambridge University Press, 2004. URL: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf (visited on Oct. 12, 2026) (cit. on pp. 15, 17, 19, 146).
- [W1] Dimitri P. Bertsekas. *Ten Simple Rules for Mathematical Writing*. Author-hosted slides, especially pages 12 and 14–28; author page https://www.mit.edu/~dimitrib/Ten_Rules.html. URL: https://web.mit.edu/dimitrib/www/Ten_Rules.pdf (visited on Oct. 12, 2026) (cit. on p. 146).

Evidence sources

- [E1] *Market Making in Sports Prediction Markets*. Interview with the founder of White Swan. Waterhouse VC. 2026. URL: <https://www.waterhousevc.com/media-archive/market-making-in-sports-prediction-markets-july-2026> (visited on Oct. 12, 2026) (cit. on p. 101).
- [E2] *Liquid prediction markets are finally here*. Announcement of institutional market making by Susquehanna International Group. Kalshi. 2024. URL: <https://news.kalshi.com/p/liquid-prediction-markets-are-finally-here> (visited on Oct. 12, 2026) (cit. on pp. 101, 102).

- [E3] *Wintermute provides liquidity on Kalshi and Polymarket*. Secondary reporting; venue identification relies on an unnamed source. Decrypt. 2026. URL: <https://decrypt.co/369475/wintermute-liquidity-kalshi-polymarket-prediction-markets> (visited on Oct. 12, 2026) (cit. on pp. 101, 102).
- [E4] *Prediction Markets Trader (job posting)*. DRW. URL: <https://www.drw.com/work-at-drw/listings/prediction-markets-trader-3332253> (visited on Oct. 12, 2026) (cit. on pp. 101, 102).
- [E5] Eric Liu. *Improving capital efficiency for parlays on prediction markets*. Current page content; the original and update chronology is ambiguous. Totalis. URL: <https://www.totalis.trade/blogs/parlay-collateral-efficiency-ilp> (visited on Oct. 12, 2026) (cit. on pp. 101, 103).
- [E6] *Totalis documentation: introduction and settlement architecture*. Totalis. URL: <https://docs.totalis.trade/> (visited on Oct. 12, 2026) (cit. on pp. 101, 103).
- [E7] Oladotun Olatunji. *Building a fully corrective Frank-Wolfe loop with a Bregman oracle*. Research preview. 2026. URL: <https://www.linkedin.com/pulse/building-fully-corrective-frank-wolfe-loop-bregman-oracle-oladotunji-ldw2c/> (visited on Oct. 12, 2026) (cit. on pp. 101, 105).
- [E8] *OpenPolyTrader repository and README*. OpenPolyTrader. URL: <https://github.com/freshtechbro/openpolytrader> (visited on Oct. 12, 2026) (cit. on pp. 101, 105).
- [E9] *OpenPolyTrader source: FwLoopEngine.ts*. OpenPolyTrader. URL: <https://github.com/freshtechbro/openpolytrader/blob/main/src/agents/projection/fw/FwLoopEngine.ts> (visited on Oct. 12, 2026) (cit. on pp. 101, 105).
- [E10] *OpenPolyTrader source: objective.ts*. OpenPolyTrader. URL: <https://github.com/freshtechbro/openpolytrader/blob/main/src/agents/projection/fw/objective.ts> (visited on Oct. 12, 2026) (cit. on pp. 101, 105).
- [E11] *OpenPolyTrader source: CP-SAT oracle service*. OpenPolyTrader. URL: <https://github.com/freshtechbro/openpolytrader/blob/main/services/ip-oracle/app.py> (visited on Oct. 12, 2026) (cit. on pp. 101, 105).
- [E12] *Official poly-market-maker repository*. Polymarket. URL: <https://github.com/Polymarket/poly-market-maker> (visited on Oct. 12, 2026) (cit. on pp. 101, 106).
- [E13] *poly-market-maker: AMM strategy documentation*. Polymarket. URL: <https://github.com/Polymarket/poly-market-maker/blob/main/docs/strategies/amm.md> (visited on Oct. 12, 2026) (cit. on p. 106).
- [E14] Mario Alves. *Automated Polymarket bot research journey*. Explicitly paper trading. LayerX. 2026. URL: <https://layerx.xyz/blog/polymarketbots> (visited on Oct. 12, 2026) (cit. on pp. 101, 106).
- [E15] *Polymarket documentation: market making, inventory, and order maintenance*. Polymarket. URL: <https://docs.polymarket.com/trading/market-making> (visited on Oct. 12, 2026) (cit. on pp. 106, 132).
- [E18] *Combos: pricing and settlement*. Kalshi Help Center. URL: <https://help.kalshi.com/en/articles/13823820-combos> (visited on Oct. 12, 2026) (cit. on p. 131).

Index

- anchor, 59
- arbitrage, 11
- bibliography, 146
- binary entropy, 20
- Bregman projection, 24
- candidate price vector, 11
- candidate trade, 49
- central limit order book, 101
- coherent price set, 11
- coherent prices, 2
- column generation, 3, 44
- common mistakes, 140
- completion criteria, 123
- constant shift, 7
- constraint generation, 136
- convex conjugate, 15
- correction objective, 14, 18
- cost function, 5
- current price, 6
- current price vector, 6
- cut generation, 134
- Dantzig–Wolfe decomposition, 3
- evidence category, 100
- extraction fraction, 51
- fail-closed oracle, 81
- formula sheet, 138
- Frank–Wolfe gap, 43
- Fully corrective Frank–Wolfe, 42
- guaranteed profit, 23
- incumbent, 54
- LCMM, 37
- LCMM constraints, 39
- ledgers, 69
- linear oracle, 42
- liquidity parameter, 7
- log likelihood, 97
- logarithmic forecast score, 19
- logarithmic market scoring rule, 7
- loss budget, 29
- marginal polytope, 11
- market-maker loss, 28
- OpenPolyTrader, 99, 105
- oracle audit, 76
- oracle interface, 111
- order book, 136
- order execution, 113
- parlay, 101
- partial outcome, 64
- Predictalot, 96
- pricing state, 5
- projection job, 114
- reading plan, 142
- reproducible logs, 115
- request for quote, 101
- restricted master, 112
- restricted-master gap, 112
- safe update, 50
- separating hyperplane, 12
- settlement
 - logical, 64
- Totalis, 99, 103
- triangle example, 34, 73
- upper gap, 54
- valid payoff set, 10
- valid payoff vector, 3
- worst-case profit, 50