Contents

Universal Quality Gates - Kleis Verification Across Domains

Status

Vision Document - How Kleis verification principles apply beyond mathematics

Core Principle

> **"Quality gates reduce volume but increase value"**

Requiring formal verification before submission/application: - Reduces volume by 50-90% (self-filtering) - Increases quality by 200-400% (only verified work submitted) - Lowers processing costs (fewer items to review) - Increases outcome value (better decisions, fewer errors)

This principle applies to **any structured domain** where: 1. Many submissions, few accepted 2. Manual review is expensive 3. Errors are costly 4. Quality matters more than quantity

---

Domain Applications

1. Grant/Funding Applications

The Problem Today NSF/NIH receives: 50,000 grant proposals/year - Review cost: $500/proposal × 50,000 = **$25M/year** - Acceptance rate: ~20% (40,000 rejections) - Common rejection reasons: - Budget doesn't add up - Timeline infeasible - Claimed results mathematically impossible - References to non-existent prior work

Pain: Reviewers waste time on nonsense proposals.

With Kleis Verification

```
structure GrantProposal {
    title: String
    budget: Budget
```

```
    timeline: Timeline
    personnel: List<Person>
    deliverables: List<Deliverable>

    // Financial axioms
    axiom budget_balance:
        budget.total = sum(budget.line_items.map( i. i.amount))

    axiom overhead_correct:
        budget.overhead = budget.direct_costs × institution.overhead_rate

    axiom salary_realistic:
         p   personnel. p.salary   max_salary(p.position, p.location)

    // Timeline axioms
    axiom deliverables_before_deadline:
         d   deliverables. d.completion_date   project_end_date

    axiom dependencies_ordered:
         d1,d2   deliverables.
             d2.depends_on(d1)   d1.completion_date < d2.start_date

    axiom timeline_realistic:
         d   deliverables. d.effort_months   available_person_months(d.period)

    // Feasibility axioms
    axiom expertise_match:
         d   deliverables.  p   personnel. p.expertise.covers(d.requirements)

    axiom prior_results_exist:
         ref   preliminary_results. verify_citation(ref) = true
}

// Submit proposal
my_proposal: GrantProposal = {...}

verify my_proposal
//   budget_balance: Off by $15,000
//   timeline_realistic: Year 2 overcommitted by 8 person-months
//   prior_results_exist: Citation [23] not found

Fix errors → re-verify →   All axioms pass → Submit
```

Impact Submissions: 50,000 → 15,000/year (**-70%**) - Self-filtering: Proposers fix errors before submission - AI-generated spam eliminated - Only serious, well-planned proposals submitted

Review cost: \$25M → \$8M/year (**-68%**) - Reviewers focus on scientific merit (not arithmetic errors) - Faster decisions (clear signal, less noise)

Acceptance rate: 20% → 40% (of submitted) - Higher absolute quality - Better use of funding

Quality: - Before: 30% of funded grants have budget/timeline issues - After: <5% have issues (mostly unforeseen circumstances)

---

2. Job Applications

The Problem Today **Large company receives:** 10,000 applications/position - HR cost: 30 min/application × 10,000 = **5,000 hours** - 99% rejected (9,900 don't meet requirements) - False claims: ~30% of resumes contain exaggerations/lies

Pain: HR drowns in unqualified applications.

With Kleis Verification

```
structure JobApplication {
    applicant: Person
    position: JobPosting
    resume: Resume
    cover_letter: Text

    // Qualification axioms
    axiom education_requirements:
        position.requires_degree
            d   applicant.education.
                d.level   position.min_degree
                verify_credential(d) = true

    axiom experience_requirements:
        applicant.years_experience   position.min_experience
         exp   applicant.experience. verify_employment(exp) = true

    axiom skill_match:
        position.required_skills   applicant.skills
         skill   applicant.skills. verify_certification(skill) = true

    axiom work_authorization:
        applicant.can_work_in(position.location) = true

    axiom salary_expectations:
        applicant.salary_expectation   position.salary_range

    axiom no_employment_gaps_unexplained:
         gap   detect_gaps(applicant.experience).
            gap.duration < 6_months   gap.explained = true

    // Verification of claims
    axiom references_real:
```

```
        ref   applicant.references. verify_contact(ref) = true

    axiom achievements_verifiable:
        achievement   applicant.achievements.
            achievement.evidence_provided = true
}

// Applicant submits
my_application: JobApplication = {...}

verify my_application
//   education_requirements: Degree claim cannot be verified
//   skill_match: "Expert in Rust" but no certification/portfolio
//   experience_requirements: Verified via LinkedIn/references
```

Impact Applications: 10,000 → 500/position (**-95%**) - Self-filtering: Unqualified don't bother applying - False claims caught before submission - AI-generated mass applications blocked

HR cost: 5,000 hours → 250 hours (**-95%**) - Review only qualified candidates - No time wasted on spam - Faster hiring cycles

Quality: - Before: 1% of applicants are great fit - After: 30% of applicants are great fit (**30× better signal**)

Verified credentials: - No more fake degrees - No more inflated experience - References actually checked

---

3. Accounting Audits

The Problem Today Public company audit: - Review 100,000 transactions manually - Cost: $500K - $2M for major audit - Time: 3-6 months - Errors still slip through (Enron, WorldCom)

Pain: Manual checking is expensive and fallible.

With Kleis Verification

```
structure FinancialStatement {
    assets: List<Asset>
    liabilities: List<Liability>
    equity: Equity
    revenue: Revenue
    expenses: Expenses

    // Fundamental accounting equation
    axiom balance_sheet:
        sum(assets.map( a. a.value)) =
        sum(liabilities.map( l. l.value)) + equity.value

    axiom revenue_recognition:
```

```
        r   revenue.
            r.recognized_date = service_delivered_date
            r.amount = contract_value

    axiom expense_matching:
        e   expenses.
            e.period = revenue_period(e.related_revenue)

    axiom depreciation_correct:
        a   assets. a.type = Fixed
            a.annual_depreciation = (a.cost - a.salvage) / a.useful_life

    axiom no_double_counting:
        t   transactions.
            count(t in all_accounts) = 1

    axiom cash_flow_reconciles:
        cash_flow.net =
            operations_cash_flow + investing_cash_flow + financing_cash_flow

    // Fraud detection axioms
    axiom benford_law:
        leading_digit_distribution(revenue.amounts)   benford_distribution
        // Deviation indicates potential fraud

    axiom related_party_disclosed:
        t   transactions. t.related_party = true   t.disclosed = true
}

// Company submits financials
acme_2024: FinancialStatement = {...}

verify acme_2024
//   balance_sheet: Assets - Liabilities - Equity = $1.2M (should be 0)
//   depreciation_correct: Building depreciation incorrect
//    benford_law: First digit distribution suspicious (possible fraud flag)
```

Impact **Audit cost:** \$1M → \$100K (**-90%**) - Automated verification handles mechanical checks - Auditors focus on judgment calls - Real-time verification (not post-quarter)

Error detection: 70% → 98% (**+40% improvement**) - Mathematical errors caught automatically - Fraud indicators flagged immediately - No more "oops, forgot to carry the 1"

Audit time: 3 months → 2 weeks - Continuous verification during quarter - Final audit just validates exceptions - Faster close process

Fraud prevention: - Benford's law violations auto-flagged - Double-counting impossible - Related party transactions must be disclosed

---

4. Research Grant Reports (NSF/NIH Progress Reports)

The Problem Annual progress reports: Often contain: - Claimed results that don't match math - Budgets that don't reconcile - Timelines that ignore dependencies - Deliverables copied from proposal (not actual progress)

Review: Program officers manually check → miss errors → bad projects continue

With Kleis

```
structure ProgressReport {
    grant: GrantProposal  // Original proposal
    period: TimeInterval
    spending: Spending
    accomplishments: List<Accomplishment>

    axiom budget_within_limit:
        spending.total   grant.budget.allocated_for(period)

    axiom spending_matches_activities:
         expense   spending.items.
             activity   accomplishments.
                 expense.related_to(activity)

    axiom deliverable_progress:
         d   grant.deliverables.
             d.due_date   period.end   d.status = Completed

    axiom results_mathematically_valid:
         result   accomplishments.
             result.formula   verify_types(result.formula) = true

    axiom personnel_effort:
         p   personnel.
             reported_effort(p) = actual_payroll(p) / p.salary
}
```

Impact: Stops fraud, ensures accountability, faster reviews.

---

5. Academic Hiring Committees

The Problem University receives: 1,000 faculty applications - Review: 3 faculty × 40 hours each = **120 hours/person** - Many unqualified applicants - CV inflation common

With Kleis

```
structure FacultyApplication {
    cv: CV
    research_statement: Text
    teaching_statement: Text
    publications: List<Publication>

    axiom publication_record:
        pub  publications.
            verify_doi(pub.doi) = true
            applicant  pub.authors

    axiom h_index_accurate:
        calculated_h_index(publications) = claimed_h_index

    axiom teaching_experience:
        years_teaching  years_since_phd + 6  // PhD + postdoc

    axiom grant_funding_real:
        grant  claimed_grants.
            verify_grant_database(grant) = true
}
```

Applications: 1,000 → 50 (**-95%**) **Review time:** 120 hours → 20 hours (**-83%**) **Quality:** Only qualified candidates get reviewed

---

6. Legal Contract Review

The Problem **Law firm reviews:** 1,000 contracts/year manually - Cost: \$500/hour × 3 hours/contract × 1,000 = **\$1.5M/year** - Errors: Boilerplate mistakes, inconsistent terms, missing clauses

With Kleis

```
structure LegalContract {
    parties: List<Party>
    terms: List<ContractTerm>
    jurisdiction: Jurisdiction

    axiom mutual_consent:
        p  parties. p.signed = true  p.capacity = Legal

    axiom consideration:
        t1,t2  terms.
            t1.benefits(party_1)  t2.benefits(party_2)

    axiom dates_consistent:
        effective_date  termination_date
```

```
        milestone ∈ terms. milestone.date ∈ [effective, termination]

    axiom jurisdiction_clauses:
        governing_law.jurisdiction = jurisdiction
        dispute_resolution.venue ∈ jurisdiction.valid_venues

    axiom mandatory_clauses_present:
        jurisdiction.required_clauses ⊆ contract.clauses
}
```

Review cost: $1.5M → $300K (**-80%**) **Errors:** 5% → 0.1% (**98% reduction**)

---

7. Insurance Claims Processing

The Problem **Insurance company:** 1M claims/year - Review cost: $50/claim × 1M = **$50M/year** - Fraud: 10% of claims (~$5B/year in US) - Processing time: 30 days average

With Kleis

```
structure InsuranceClaim {
    policy: Policy
    incident: Incident
    claimed_amount: Currency

    axiom policy_active:
        incident.date ∈ [policy.start_date, policy.end_date]

    axiom covered_peril:
        incident.type ∈ policy.covered_perils

    axiom within_limits:
        claimed_amount ≤ policy.coverage_limit

    axiom deductible_applied:
        payout = max(0, claimed_amount - policy.deductible)

    axiom no_duplicate_claim:
        ¬ ∃ other_claim. other_claim.incident_id = this.incident_id

    axiom timeline_plausible:
        incident.date ≤ claim.filing_date ≤ incident.date + 30_days

    // Fraud detection
    axiom amounts_reasonable:
        claimed_amount ∈ reasonable_range(incident.type, incident.severity)

    axiom pattern_detection:
```

```
        claimant.claim_frequency < suspicious_threshold
}
```

Impact: - **Auto-approve:** 60% of claims (pass all axioms instantly) - **Flag for review:** 30% (axiom warnings) - **Auto-reject:** 10% (clear violations, likely fraud)

Cost: $50M → $10M (**-80%**) **Fraud:** $5B → $500M (**-90%**) **Processing time:** 30 days → 1 day for auto-approved

---

8. Building Permits

The Problem **City receives:** 10,000 permit applications/year - Review: 2 weeks/permit average - Common issues: Code violations, missing calculations, unsafe designs - Cost: City employs 50 reviewers full-time

With Kleis

```
structure BuildingPermit {
    design: StructuralDesign
    location: Address
    zoning: ZoningDistrict

    axiom zoning_compliance:
        design.height   zoning.max_height
        design.setback   zoning.min_setback
        design.use   zoning.permitted_uses

    axiom structural_safety:
         member   design.structural_members.
            stress(member, design.loads)   member.material.yield_strength / safety_factor

    axiom load_calculations:
        design.live_load   building_code.min_live_load(design.use)
        design.dead_load = sum(design.components.map( c. c.weight))

    axiom egress_requirements:
        design.exits.count   required_exits(design.occupancy)
         exit   design.exits. exit.width   code.min_exit_width

    axiom accessibility:
        design.ada_compliant = true
}
```

Impact: - **Auto-approve:** 40% instantly (all code requirements met) - **Expedited review:** 40% (minor issues flagged) - **Full review:** 20% (complex/novel designs)

Staffing: 50 reviewers → 15 reviewers (**-70%**) **Approval time:** 2 weeks → 2 days for auto-approved

---

9. Tax Returns (IRS) - And Tax Laws Written in Kleis!

The Revolutionary Insight Instead of checking returns against natural-language tax code, write the tax code ITSELF in Kleis.

The Problem with Current Tax Laws U.S. Tax Code: 70,000+ pages of natural language - Ambiguous phrasing → different interpretations - Loopholes from unclear wording - Expensive tax lawyers needed to interpret - IRS and taxpayers often disagree - Courts decide what laws "really mean" - Software bugs from misreading code

Example ambiguity:

```
"Ordinary and necessary business expenses may be deducted"
  - What is "ordinary"?
  - What is "necessary"?
  - Courts decide case-by-case
```

Tax Code in Kleis (Executable Law)

```
// U.S. Tax Code (Kleis Edition)
// Title 26, Section 162(a) - Business Expenses

structure BusinessExpense {
    amount: Currency
    category: ExpenseCategory
    business: Business

    axiom ordinary_expense:
        category   industry_standard_expenses(business.industry)
        amount   industry_median(category, business.size) × 2.0

    axiom necessary_expense:
        expense.directly_related_to(business.revenue_activities) = true

    axiom deductible_calculation:
        deductible_amount = amount × business_use_percentage

    // Specific exclusions (unambiguous)
    axiom not_capital_expenditure:
        useful_life(expense) < 1_year   amount < 2500

    axiom not_personal:
        expense.personal_benefit_percentage < 0.1
}

// Standard deduction (2024 example)
axiom standard_deduction_2024:
```

```
    standard_deduction(Single) = 14600
    standard_deduction(MarriedFilingJointly) = 29200
    standard_deduction(HeadOfHousehold) = 21900

// Tax brackets (2024)
operation tax_liability : (Currency, FilingStatus) → Currency
    // No ambiguity - exact formula
    match (status, taxable_income) {
        (Single, income) if income   11600 => income × 0.10,
        (Single, income) if income   47150 => 1160 + (income - 11600) × 0.12,
        (Single, income) if income   100525 => 5426 + (income - 47150) × 0.22,
        // ... etc
    }

structure TaxReturn {
    taxpayer: Person
    income: Income
    deductions: List<Deduction>
    credits: List<Credit>

    // AUTOMATICALLY VERIFIED AGAINST TAX CODE AXIOMS

    axiom income_matches_w2:
        income.wages = sum(taxpayer.w2_forms.map( w. w.wages))

    axiom deduction_limits:
         d   deductions.
            // Direct reference to tax code axioms!
            verify(d, BusinessExpense)   verify(d, CharitableDonation)   ...

    axiom standard_or_itemized:
        deductions.type = Standard   deductions.type = Itemized

    axiom tax_calculation:
        // References executable tax_liability function from tax code
        tax = tax_liability(taxable_income, filing_status) - credits
}
```

The Transformation Current state:

```
Tax Code (natural language)
  ↓ [interpreted by]
IRS regulations (more text)
  ↓ [interpreted by]
Tax software developers (bugs possible)
  ↓ [interpreted by]
Tax preparers (errors possible)
  ↓
```

```
Taxpayer return
  ↓ [checked manually]
IRS auditor
```

With Kleis:

```
Tax Code (executable Kleis)
  ↓ [IS the implementation]
Taxpayer return (Kleis)
  ↓ [automatic verification]
Verified  or Errors flagged
```

Zero interpretation needed - the code IS the law.

Benefits: Tax Code as Executable Specification 1. No Ambiguity - "Ordinary and necessary" → precise algorithmic definition - Courts don't need to interpret (it's unambiguous) - Everyone sees same rules (transparent)

2. Instant Compliance Checking

```
// Citizen writes their return
my_return: TaxReturn = {
    income: 75000,
    deductions: [business_expense(500, "office supplies")],
    ...
}

// Instant verification against tax code
verify my_return against us_tax_code_2024
//   business_expense(500, "office supplies"):
//     Not ordinary for your industry (software, remote work)
//     Did you mean "home office equipment"?
```

3. No Tax Software Bugs - TurboTax, H&R Block just *call* Kleis tax code - Implementation = law (by definition correct) - No more "software calculated wrong amount"

4. Version Control for Laws

```
git diff us_tax_code_2023.kleis us_tax_code_2024.kleis

+ axiom energy_credit_2024:
+     solar_installation.credit_rate = 0.30
- axiom energy_credit_2023:
-     solar_installation.credit_rate = 0.26

# Citizens see EXACTLY what changed
```

5. Impact Analysis Before Passing Laws

```
// Proposed change: Increase standard deduction
axiom proposed_standard_deduction_2024:
    standard_deduction(Single) = 16000  // Up from 14600
```

```
// Simulate impact on 150M returns
impact = simulate(tax_returns_2023, proposed_change)
// Result:
//   Revenue loss: $45B
//   Affected taxpayers: 85M
//   Average savings: $529/taxpayer
//   Distribution by income bracket: [chart]

// Congress sees EXACTLY what law does before voting
```

6. Eliminates Tax Court Ambiguity

```
Current: "Is home office deduction ordinary and necessary?"
         → Tax Court decides
         → Years of litigation
         → Precedent only

Kleis:   "Does this expense satisfy BusinessExpense axioms?"
         → Verify → Answer is yes or no
         → Instant
         → No litigation needed (unambiguous)
```

Impact on Society **IRS:** - Processing: $4B → $400M (**-90%**) - Fraud: $50B → $5B (**-90%**) - Audit staff: 75,000 → 15,000 (**-80%**) - **Saves:** $3.6B/year + $45B in fraud

Taxpayers: - Instant verification (not 21-day wait) - No ambiguity (know exactly what's allowed) - No expensive tax preparers for simple returns - Can verify H&R Block got it right - **Saves:** ~$10B/year in preparation fees

Congress: - Impact analysis before passing laws - No unintended consequences - Version-controlled legislation - Transparent to citizens

Tax Attorneys: - Less work on interpretation - More work on legitimate optimization - Focus on intent, not ambiguity

Political Feasibility **Why Congress would adopt this:** - Transparency demands (citizens can read executable code) - Cost savings (IRS efficiency) - Fraud reduction (huge political win) - Constituent benefit (faster refunds) - Eliminates "gotcha" taxes (rules are clear)

Objections: - "Lawyers make money from ambiguity" → True, but voters want clarity - "Tax code too complex for code" → Kleis handles complexity better than natural language - "Special interests like loopholes" → Kleis makes loopholes obvious (sunlight is disinfectant)

Precedent: Estonia **Estonia already has:** - 95% of tax returns filed electronically - Pre-filled returns (government has data) - Average filing time: 3 minutes

With Kleis, Estonia could: - Make tax code executable - Instant verification - Zero interpretation needed - Model for other countries

---

Impact: 150M Tax Returns/Year

Current: - 150M returns × $50 processing = $7.5B/year - 30% have errors requiring correction - 45M people pay preparers $200 avg = $9B/year - IRS audit cost: $300M/year - **Total cost to society:** ~$17B/year

With Kleis Tax Code: - Auto-processing: 95% of returns (**-90% manual review**) - Errors: 2% (caught before submission) - Preparation: DIY (software = law, unambiguous) - Audits: Automated for simple cases - **Total cost:** ~$2B/year

Net savings: $15B/year to American society

The Universal Pattern

---

10. Loan Applications

The Problem Bank receives: 100,000 mortgage applications/year - Underwriting cost: $500/app × 100K = **$50M/year** - Default rate: 3% (often due to mis-stated income/debt)

With Kleis

```
structure MortgageApplication {
    applicant: Borrower
    property: Property
    loan_amount: Currency

    axiom debt_to_income:
        (applicant.monthly_debt + loan.monthly_payment) / applicant.monthly_income   0.43

    axiom loan_to_value:
        loan_amount / property.appraised_value   0.80

    axiom income_verified:
         income_source   applicant.income.
            verify_w2_or_paystub(income_source) = true

    axiom credit_score:
        applicant.credit_score   620  // Conventional loan minimum

    axiom employment_stable:
        applicant.current_employment.duration   2_years
        applicant.same_field_employment.duration   5_years

    axiom down_payment_sourced:
        down_payment.amount   0.20 × property.price
         source   down_payment.sources.
```

```
            verify_bank_statements(source) = true
}
```

Impact: - **Instant decision:** 70% (clear approve/deny) - **Manual review:** 30% (edge cases) - **Processing time:** 30 days → 1 day - **Default rate:** 3% → 0.5% (**-83%** via better screening)

---

9. Travel Reimbursements

The Problem **Large organization:** 50,000 employees × 4 trips/year = **200,000 reimbursements** - Review: 20 min/request × 200K = **67,000 hours/year** - Common errors: No receipts, over policy limits, duplicate submissions, math errors - Fraud: ~5% of claims have inflated/fake expenses

With Kleis

```
structure TravelReimbursement {
    employee: Employee
    trip: TripDetails
    expenses: List<Expense>
    receipts: List<Receipt>

    axiom receipts_required:
        e  expenses. e.amount > 25  r  receipts. r.matches(e)

    axiom within_policy_limits:
        e  expenses.
           e.amount  policy.max_amount(e.category, e.location)

    axiom trip_authorized:
        trip.approval_date < trip.start_date
        trip.approver.can_authorize(trip.cost)

    axiom dates_consistent:
        e  expenses. e.date  [trip.start_date, trip.end_date + 1_day]

    axiom math_correct:
        claimed_total = sum(expenses.map( e. e.amount))

    axiom no_duplicates:
        ¬ e1,e2  expenses. e1.receipt_id = e2.receipt_id

    axiom mileage_reasonable:
        e  expenses. e.type = Mileage
           e.miles  map_distance(e.origin, e.destination) × 1.2

    axiom per_diem_or_meals:
        ¬(claimed_per_diem  e  expenses. e.type = Meal)
```

```
        // Cannot claim both
}
```

Impact: - **Auto-approve:** 80% (all rules satisfied, receipts attached) - **Flag for review:** 15% (missing receipts, over limit) - **Reject:** 5% (clear fraud indicators)

Processing time: 2 weeks → 1 day **Cost:** 67K hours → 10K hours (**-85%**) **Fraud:** 5% → 0.5% (**-90%**)

---

10. FAFSA (Student Financial Aid)

The Problem US Department of Education: 17 million FAFSA applications/year - Processing: Complex verification, income checks, dependency status - Errors: 30% of applications have mistakes requiring correction - Fraud: $500M/year in improper aid disbursement - Time: 3-6 weeks for processing

With Kleis

```
structure FAFSAApplication {
    student: Student
    family: FamilyInfo
    income: IncomeInfo
    assets: AssetInfo

    // Dependency status axioms
    axiom dependency_age:
        student.age   24   student.status = Independent

    axiom dependency_married:
        student.marital_status = Married   student.status = Independent

    axiom dependency_military:
        student.military_service = true   student.status = Independent

    // Income verification axioms
    axiom income_matches_tax:
        family.reported_income = family.tax_return.agi
        verify_irs_data(family.tax_return) = true

    axiom asset_reporting:
        family.net_worth = sum(assets) - sum(liabilities)

    // EFC calculation axioms
    axiom efc_formula:
        expected_family_contribution =
            federal_methodology(family.income, family.assets, family.size)

    axiom pell_eligibility:
```

```
        efc < pell_threshold   eligible_for_pell = true

    axiom loan_limits:
        requested_loans   federal_loan_limit(student.year, student.dependency)

    // Consistency checks
    axiom family_size_reasonable:
        family.household_size =
            count(family.members.filter( m. m.supported_by_student_family))

    axiom school_attendance:
        student.enrolled_at_least_half_time = true

    // Fraud prevention
    axiom income_pattern_normal:
        family.income_this_year   family.income_last_year ± 0.5
          family.income_change_explained = true
}
```

Impact: - **Auto-process:** 11M applications (**65%** - all checks pass) - **Simple verification:** 5M (**30%** - minor issues, quick fix) - **Full review:** 1M (**5%** - complex situations)

Processing time: 3-6 weeks → 1 day for verified **Error rate:** 30% → 2% (**-93%**) **Fraud:** $500M/year → $50M/year (**-90%**) **Staff savings:** 1,000 processors → 200 (**-80%**)

Student benefit: - Know eligibility instantly (not 6 weeks later) - Clear error messages (not vague rejections) - Auto-correction suggestions - Faster aid disbursement

---

11. Government Benefits Applications

Unemployment Insurance, Food Stamps, Medicaid US receives: ~80M benefit applications/year across programs - Processing cost: $50/app × 80M = **$4B/year** - Error rate: 10-15% (improper payments) - Fraud: $10B/year across all programs

```
structure BenefitApplication {
    applicant: Person
    benefit_type: BenefitProgram
    household: HouseholdInfo

    axiom income_eligibility:
        household.income   benefit_type.income_threshold(household.size)

    axiom asset_limits:
        household.countable_assets   benefit_type.asset_limit

    axiom citizenship_or_status:
        applicant.citizenship = USCitizen
        applicant.status   benefit_type.eligible_statuses
```

```
    axiom work_requirements:
        benefit_type.requires_job_search
            applicant.job_search_activities   benefit_type.min_activities

    axiom no_double_dipping:
        ¬ other_benefit.
            other_benefit.type = this.benefit_type
            other_benefit.active = true

    axiom household_composition_verified:
         member   household.members.
            verify_residency(member, household.address) = true
}
```

Impact: - Processing: $4B → $800M (**-80%**) - Errors: 15% → 1% (**-93%**) - Fraud: $10B → $1B (**-90%**) - Speed: 30 days → 1 day

---

12. Visa Applications

The Problem **US receives:** 10M visa applications/year - Processing: 2-12 months - Denial rate: 30% (often due to incomplete/incorrect docs) - Review: Consular officers manually check each

```
structure VisaApplication {
    applicant: ForeignNational
    visa_type: VisaCategory
    documents: List<Document>

    axiom required_documents:
        visa_type.required_docs   documents.map( d. d.type)

    axiom passport_valid:
        applicant.passport.expiry > visa_type.duration + 6_months

    axiom financial_support:
        visa_type.requires_financial_proof
            applicant.bank_balance   visa_type.min_funds
            sponsor.guarantees_support = true

    axiom ties_to_home:
        visa_type = Tourist
            (applicant.owns_property_in_home
             applicant.has_job_in_home
             applicant.has_family_in_home)

    axiom no_overstay_risk:
        applicant.previous_visa_compliance = true
```

```
    axiom background_clear:
        ¬ violation   disqualifying_violations.
            applicant.history.contains(violation)
}
```

Impact: - **Instant decision:** 40% (clear approve/deny) - **Interview required:** 30% (need human judgment) - **Rejected before interview:** 30% (clear disqualifications)

Processing time: 6 months → 1 week for verified **Consular officer workload:** -60%

---

13. Network Protocols & Firewall Rules

The Problem Enterprise networks: - 1,000+ firewall rules per organization - Rules written by hand, applied manually - 60% of breaches due to misconfigured firewalls - Average: 20% of firewall rules are redundant/conflicting - Configuration errors cause 95% of network outages

Protocol implementations: - RFC specifications in natural language → misinterpretations - Packet validation done ad-hoc → vulnerabilities - Routing policies complex → misconfigurations - No formal verification before deployment

Pain: - Security breaches from misconfigured rules - Network outages from conflicting policies - Packet injection attacks from improper validation - Routing loops from policy errors

With Kleis: Network Protocol Verification

```
// IPv4 Packet Structure (RFC 791)
structure IPv4Packet {
    version: Nat            // Must be 4
    ihl: Nat                // Internet Header Length (5-15)
    dscp: Nat               // Differentiated Services
    ecn: Nat                // Explicit Congestion Notification
    total_length: Nat       // Total packet length
    identification: Nat     // Fragment identification
    flags: Flags            // DF, MF flags
    fragment_offset: Nat    // Fragment position
    ttl: Nat                // Time to live (1-255)
    protocol: Protocol      // TCP=6, UDP=17, ICMP=1, etc.
    checksum: Nat           // Header checksum
    source_ip: IPv4Address
    dest_ip: IPv4Address
    options: List<IPv4Option>
    payload: Bytes

    // RFC 791 Axioms - Packet Format Validation
    axiom version_is_four:
        version = 4

    axiom header_length_valid:
```

```
        ihl   5   ihl   15
        ihl × 4 = header_bytes.length

    axiom total_length_consistent:
        total_length = (ihl × 4) + payload.length
        total_length   65535

    axiom ttl_nonzero:
        ttl > 0   ttl   255

    axiom checksum_valid:
        checksum = compute_ipv4_checksum(header_bytes)

    axiom fragment_rules:
        (flags.MF = false   fragment_offset = 0)    // Not fragmented
        (flags.MF = true   fragment_offset > 0)      // Fragmented

    axiom options_length:
        ihl > 5   options.length = (ihl - 5) × 4

    // Security axioms
    axiom no_spoofed_source:
        ¬is_private(source_ip)   from_internal_network = true

    axiom no_martian_addresses:
        ¬is_martian(source_ip)   ¬is_martian(dest_ip)
        // Martian: 0.0.0.0/8, 127.0.0.0/8, 224.0.0.0/4, etc.
}

// TCP Packet (RFC 793)
structure TCPSegment {
    ip_packet: IPv4Packet
    source_port: Port
    dest_port: Port
    sequence_number: Nat
    ack_number: Nat
    data_offset: Nat
    flags: TCPFlags        // SYN, ACK, FIN, RST, PSH, URG
    window_size: Nat
    checksum: Nat
    urgent_pointer: Nat
    options: List<TCPOption>
    data: Bytes

    axiom protocol_is_tcp:
        ip_packet.protocol = TCP  // 6

    axiom data_offset_valid:
```

```
        data_offset   5   data_offset   15
        data_offset × 4 = tcp_header_bytes.length

    axiom checksum_valid:
        checksum = compute_tcp_checksum(ip_packet, tcp_segment)

    axiom flag_consistency:
        // Can't have SYN and FIN simultaneously
        ¬(flags.SYN   flags.FIN)
        // ACK must be set if not first SYN
        (flags.SYN   ack_number = 0)   flags.ACK

    axiom window_size_reasonable:
        window_size   65535

    axiom sequence_number_ordering:
        is_established_connection
            sequence_number > previous_sequence_number
}
```

Firewall Rule Verification

```
structure FirewallRule {
    rule_id: Nat
    priority: Nat
    source_ip: IPRange
    dest_ip: IPRange
    source_port: PortRange
    dest_port: PortRange
    protocol: Protocol
    action: Action         // ALLOW, DENY, REJECT
    direction: Direction   // INBOUND, OUTBOUND
    interface: Interface

    // Rule validity axioms
    axiom ip_ranges_valid:
        is_valid_cidr(source_ip)   is_valid_cidr(dest_ip)

    axiom port_ranges_valid:
        source_port.min   source_port.max
        dest_port.min   dest_port.max
        source_port.max   65535
        dest_port.max   65535

    axiom protocol_port_consistency:
        protocol = ICMP
            (source_port = ANY   dest_port = ANY)
}
```

```
structure FirewallPolicy {
    rules: List<FirewallRule>
    default_action: Action

    // Policy consistency axioms
    axiom no_duplicate_rules:
        r1,r2  rules. r1.rule_id  r2.rule_id
            ¬rules_match_same_traffic(r1, r2)

    axiom no_shadowed_rules:
        r1,r2  rules.
            r1.priority < r2.priority
            r1.matches_superset_of(r2)
                r1.action = r2.action  // Otherwise r2 never applies

    axiom no_conflicting_rules:
        r1,r2  rules.
            r1.matches_same_as(r2)  r1.action = r2.action

    axiom explicit_deny_before_allow:
        r_deny,r_allow  rules.
            r_deny.action = DENY  r_allow.action = ALLOW
            r_deny.specificity > r_allow.specificity
                r_deny.priority < r_allow.priority

    axiom no_overly_permissive_rules:
        r  rules. r.action = ALLOW
            ¬(r.source_ip = ANY  r.dest_ip = ANY
              r.dest_port = ANY  r.protocol = ANY)

    // Security best practices
    axiom block_known_malicious_ports:
        r  rules.
            r.dest_port  known_malicious_ports
                r.action = DENY

    axiom rate_limiting_on_public:
        r  rules.
            r.source_ip = ANY  r.direction = INBOUND
                r.has_rate_limit = true
}
```

Routing Protocol Verification

```
structure BGPRoute {
    prefix: IPPrefix
    next_hop: IPv4Address
```

```
    as_path: List<ASNumber>
    local_pref: Nat
    med: Nat                // Multi-Exit Discriminator
    community: List<Community>

    // BGP validity axioms
    axiom prefix_valid:
        prefix.length  8  prefix.length  32

    axiom no_as_path_loops:
         as  as_path. count(as in as_path) = 1

    axiom next_hop_reachable:
        is_directly_connected(next_hop)
        exists_route_to(next_hop)

    axiom no_bogon_prefixes:
        ¬is_bogon(prefix)  // 0.0.0.0/8, 10.0.0.0/8, etc.

    axiom as_path_not_too_long:
        as_path.length  255

    axiom local_pref_for_ibgp_only:
        is_ibgp_session  local_pref  null
}

structure RoutingTable {
    routes: List<Route>
    default_route: Route

    axiom no_routing_loops:
         r  routes. ¬creates_loop(r, routes)

    axiom most_specific_wins:
         r1,r2  routes.
            r1.prefix.contains(r2.prefix)
                r2.prefix.length > r1.prefix.length
                r2 = r1

    axiom next_hop_consistency:
         r  routes.
            r.next_hop  null
                 interface. can_reach(r.next_hop, interface)
}
```

Network Configuration Verification

```
structure NetworkConfiguration {
    interfaces: List<NetworkInterface>
    firewall_policy: FirewallPolicy
    routing_table: RoutingTable
    dns_servers: List<IPv4Address>
    ntp_servers: List<IPv4Address>

    // Network-wide consistency axioms
    axiom no_ip_conflicts:
        i1,i2  interfaces.
           i1  i2  i1.ip_address  i2.ip_address

    axiom subnet_consistency:
        i1,i2  interfaces.
           same_subnet(i1.ip_address, i2.ip_address)
               i1.subnet_mask = i2.subnet_mask

    axiom default_gateway_reachable:
        i  interfaces.
           routing_table.default_route.next_hop  i.subnet

    axiom dns_servers_reachable:
        dns  dns_servers.
            route  routing_table. route.can_reach(dns)

    axiom firewall_allows_essential_services:
        allows_dns(firewall_policy)
        allows_ntp(firewall_policy)
        allows_management(firewall_policy)

    // Security axioms
    axiom management_interface_protected:
        i  interfaces. i.type = Management
           firewall_policy.blocks_external_access(i)

    axiom no_promiscuous_mode:
        i  interfaces. i.promiscuous_mode = false
}
```

Real-World Example: Web Server Deployment

```
// Define what a secure web server config looks like
structure WebServerDeployment {
    server_ip: IPv4Address
    https_port: Port        // 443
    http_port: Port         // 80 (redirect only)
    firewall: FirewallPolicy
    network: NetworkConfiguration
```

```
    axiom https_port_open:
        rule  firewall.rules.
            rule.dest_ip = server_ip
            rule.dest_port = https_port
            rule.protocol = TCP
            rule.action = ALLOW

    axiom http_redirect_only:
        rule  firewall.rules.
            rule.dest_port = http_port
            rule.action = ALLOW
        // But application must redirect HTTP → HTTPS

    axiom no_plain_http_traffic:
        rule  firewall.rules.
            rule.dest_port = http_port
            rule.action = ALLOW
                application_redirects_to_https = true

    axiom management_ports_blocked:
        port  [22, 3389, 5900].  // SSH, RDP, VNC
            rule  firewall.rules.
                rule.dest_ip = server_ip
                rule.dest_port = port
                rule.source_ip = ANY
                    rule.action = DENY

    axiom rate_limiting_enabled:
        rule  firewall.rules.
            rule.dest_ip = server_ip
            rule.dest_port = https_port
                rule.rate_limit  1000  // requests/second

    axiom ddos_protection:
        firewall.has_syn_flood_protection = true
        firewall.has_connection_limit = true
}

// Deploy and verify
my_deployment: WebServerDeployment = {
    server_ip: 203.0.113.42,
    https_port: 443,
    http_port: 80,
    firewall: my_firewall,
    network: my_network
}
```

```
verify my_deployment
//   management_ports_blocked: SSH (port 22) is open to 0.0.0.0/0
//   rate_limiting_enabled: No rate limit on HTTPS rule
//   https_port_open: Verified
//   ddos_protection: Verified
```

Impact: Network Security & Operations **Firewall misconfigurations:** - **Before:** 60% of breaches due to misconfigured firewalls - **After:** Pre-deployment verification catches 95% of errors - **Impact:** Fewer security incidents, faster deployment

Network outages: - **Before:** 95% of outages due to configuration errors - **After:** Routing loops, IP conflicts caught before deployment - **Impact:** 10× reduction in network downtime

Protocol implementation bugs: - **Before:** Packet validation bugs → vulnerabilities - **After:** RFC compliance verified automatically - **Impact:** Fewer CVEs, more secure implementations

Operational efficiency: - **Before:** Manual review of firewall rules (hours/days) - **After:** Instant verification (seconds) - **Impact:** Deploy confidently, iterate faster

Compliance: - **Before:** Manual audits of network security - **After:** Provably compliant configurations - **Impact:** Pass audits automatically

Market Opportunity **Network security market:** - Firewall management: $5B/year - Network configuration tools: $3B/year - Security auditing: $2B/year - **Total:** ~$10B/year

Kleis value: - Verify firewall rules before deployment - Catch configuration errors automatically - Prove RFC compliance for protocol implementations - Reduce security incidents by 60-90%

The Universal Pattern

Formula for Any Domain

```
1. Define structure with axioms (what makes submission valid)
2. Require verification before submission
3. Self-filtering effect (people fix errors locally)
4. Volume drops 50-95%
5. Quality increases 200-400%
6. Review cost drops 70-95%
7. Better outcomes (fewer errors, fraud, waste)
```

Examples Across Domains

Domain	Volume Change	Quality Change	Cost Savings	Primary Benefit
arXiv papers	-85%	+325%	$1.7M/year	Scientific integrity
Grant proposals	-70%	+300%	$17M/year	Better funded research

Domain	Volume Change	Quality Change	Cost Savings	Primary Benefit
Job applications	-95%	+3000%	$2M/position	Find right talent
Accounting audits	-50%	+400%	$1M/audit	Fraud prevention
Tax returns	-5%	+500%	$285M/year	Faster refunds
Loan applications	-30%	+400%	$35M/year	Lower defaults
Building permits	-60%	+300%	$10M/year	Faster approvals
Insurance claims	-40%	+500%	$40M/year	Fraud reduction
Network security	-90%	+1000%	$5B/year	Breach prevention

Universal principle: Quality gates reduce noise, amplify signal.

Why This Works: Game Theory

The Submission Game

Without verification (current state):

```
Submitter strategy: "Submit everything, hope something passes"
  - Cost to submit: Low (just time to fill forms)
  - Cost to review: High (reviewers check manually)
  - Outcome: Many low-quality submissions
```

With verification (Kleis):

```
Submitter strategy: "Only submit if verification passes"
  - Cost to submit: Higher (must pass verification first)
  - Cost to review: Low (only verified submissions)
  - Outcome: Few high-quality submissions
```

The Equilibrium Shift

Before: - **Submitters:** Low cost → submit garbage - **Reviewers:** High cost → overwhelmed - **Result:** Low-quality equilibrium

After: - **Submitters:** High cost locally → fix errors before submit - **Reviewers:** Low cost → manageable load - **Result:** High-quality equilibrium

The trick: Transfer verification cost from reviewer to submitter, but make it **automated** (so submitters don't mind).

Implementation Strategy

Phase 1: Mathematics (Current)

- Prove concept with arXiv papers
- Show quality-volume trade-off works
- Build credibility

Phase 2: Adjacent Domains

- Grant proposals (NSF/NIH pilot)
- Academic hiring (university pilot)
- Demonstrate 80%+ cost savings

Phase 3: Financial Sector

- Accounting audits (Big 4 pilot)
- Tax returns (IRS partnership)
- Huge ROI ($billions saved)

Phase 4: Universal Platform

- Standard library for all domains
- Open-source verification engine
- API for integration with any submission system
- **Kleis becomes infrastructure for quality**

Market Opportunity

Total Addressable Market (TAM)

Just in verification costs: - Academic grants: $25M/year (US alone) - Job screening: $10B/year (US corporate hiring) - Accounting audits: $20B/year (global Big 4) - Tax processing: $5B/year (IRS + preparers) - Insurance claims: $50B/year (processing + fraud) - Network security: $10B/year (firewall management + auditing) - **Total:** ~$95B/year in manual review costs

Kleis value proposition: Reduce these costs by 70-90% via automated verification.

Revenue Models

Option 1: SaaS Platform - Kleis Cloud hosting - Per-verification pricing - Enterprise licenses

Option 2: Open-Source + Services - Free verification engine - Revenue from consulting, training, custom domains

Option 3: Institutional Partnerships - arXiv, NSF, IRS license Kleis - One-time integration fee + annual support

---

Social Impact

Access to Justice

- **Legal document verification** democratized
- Small businesses can verify contracts without $500/hour lawyers
- "Was my contract valid?" → Kleis checks instantly

Financial Inclusion

- **Loan application verification** transparent
- Applicants see exactly why they were denied
- No more "black box" decisions

Scientific Integrity

- **Mathematics verification** accessible to all
- Developing world researchers can verify work
- Level playing field (not just well-funded labs)

Government Efficiency

- **Tax returns, permits, benefits** processed instantly
- Citizens get faster service
- Government saves billions

Challenges & Risks

Challenge 1: "Nobody Will Adopt This"

Counter: - Start with optional verification (carrots) - Demonstrate clear ROI (cost savings) - Build credibility with early wins (arXiv pilot) - Eventually becomes standard (sticks)

Challenge 2: "It's Too Restrictive"

Counter: - Verification catches real errors (helpful, not bureaucratic) - Errors fixed locally (not rejection surprise) - Improves work quality (side effect is submission reduction) - Users appreciate quality bar

Challenge 3: "What About Edge Cases?"

Counter: - Manual review option always available - Start with 80% coverage (common cases) - Expand axioms over time - Edge cases = opportunity to improve system

Challenge 4: "Privacy Concerns"

Counter: - Verification is structural (checks math/logic, not content) - Can run locally before submission - Open-source engine (transparent, auditable) - No data retention beyond verification result

---

Roadmap

2025-2026: Mathematics Proof-of-Concept

- arXiv integration
- Demonstrate quality-volume trade-off
- Publish results

2027-2028: Academic Expansion

- NSF/NIH grant proposals
- University hiring
- Show billions in savings

2029-2030: Financial Sector

- Accounting audits (Big 4)
- Tax returns (IRS)
- Massive ROI demonstrated

2031-2035: Universal Platform

- 50+ domains covered
- Standard verification infrastructure
- **Kleis = quality layer for submissions everywhere**

Vision Statement

By 2035, "Kleis-Verified" will be the quality standard across all structured domains.

Just as: - SSL/TLS secures the web - Git version-controls code - Docker containers isolate software

Kleis verifies structured submissions: - Academic papers - Grant applications - Legal contracts - Financial statements - Government forms - Network protocols and firewall rules - **Any domain with rules**

The goal: Make quality gates automated, accessible, and universal. Reduce noise, amplify signal, improve outcomes everywhere.

Status: Vision documented. Implementation starts with mathematics (arXiv), expands systematically.

Impact: Potential to save $100B+/year globally in review costs while dramatically improving decision quality across society. Network security alone represents $5B/year in breach prevention.

Timeline: 10-year vision, starting now with type system implementation.