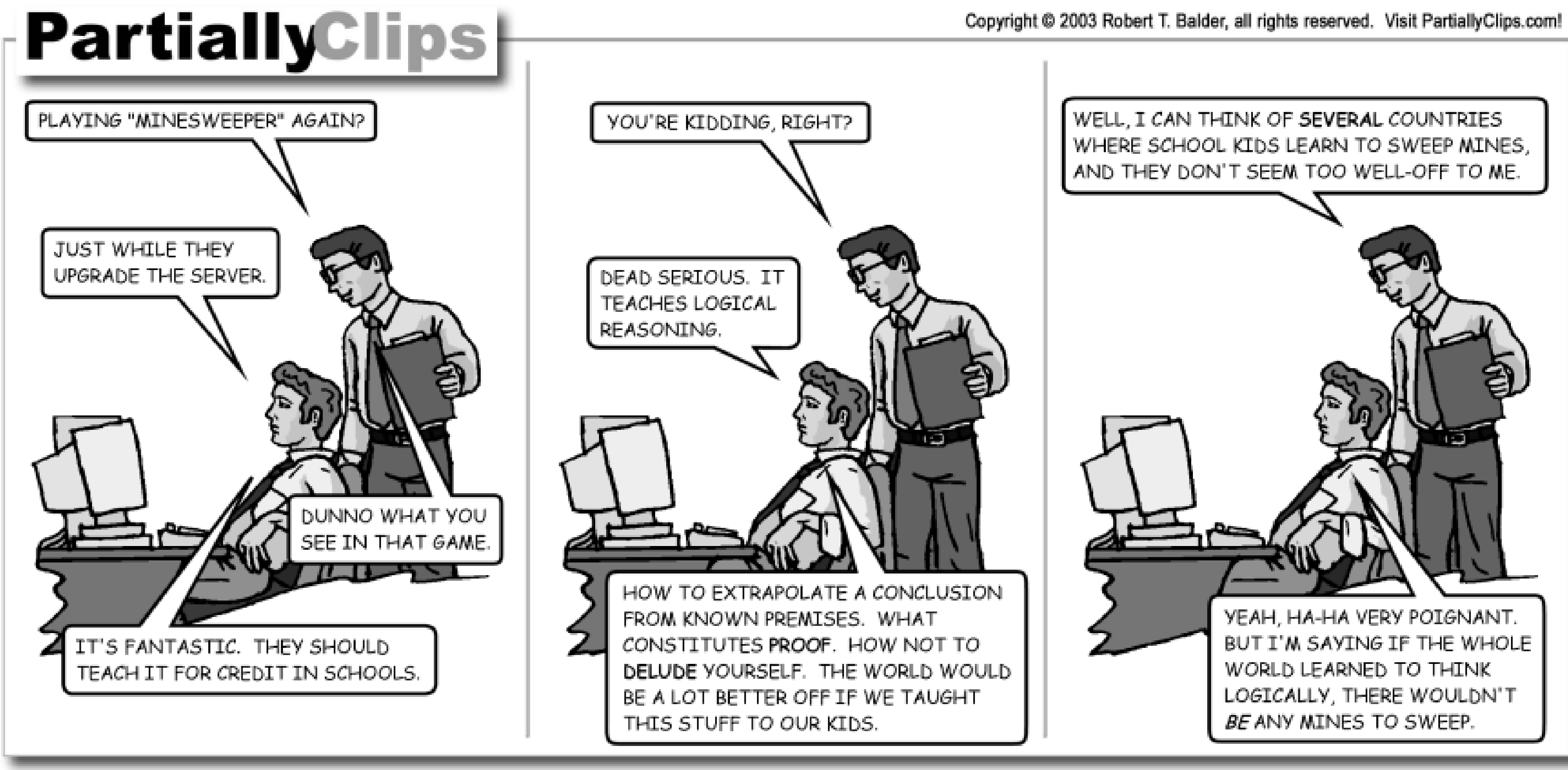
Automated Theorem Proving and Proof Checking



Noninterference proof (cont'd)

- Operational semantics:
  1. Machine semantics: $(\sigma, C) \hookrightarrow (\sigma', C')$
  2. Security-aware, intermediate semantics: $(\sigma, C) \underset{L,D}{\rightarrow} (\sigma', C')$
- Safety:
  - if $\vdash \{P\}C\{Q\}$, then C runs safely under semantics 1 on any initial state satisfying P
- Security:
  - if $\mathbf{D} \vdash_L \{P\}C\{Q\}$, then C runs safely under semantics 2 on any initial state satisfying P

Cunning Theorem-Proving Plan

- There are full-semester courses on automated deduction; we will elide details.
- Logic Syntax
- Theories
- Satisfiability Procedures
- Mixed Theories
- Theorem Proving
- Proof Checking
- SAT-based Theorem Provers (cf. Engler paper)

One-Slide Summary

- An **automated theorem prover** mechanically performs deduction, producing proofs for true propositions.
- Theorem provers are built atop **decision procedures** for individual **theories** (e.g., theory of arithmetic, theory of uninterpreted functions).
- Two common theorem prover architectures are **Cooperating Decision Procedures**, which broadcast discovered equalities, and **SAT-Based Theorem Provers**, which use SAT solvers to decompose the problem.
- **Proof Checking** is equivalent to type checking in a dependent type system.

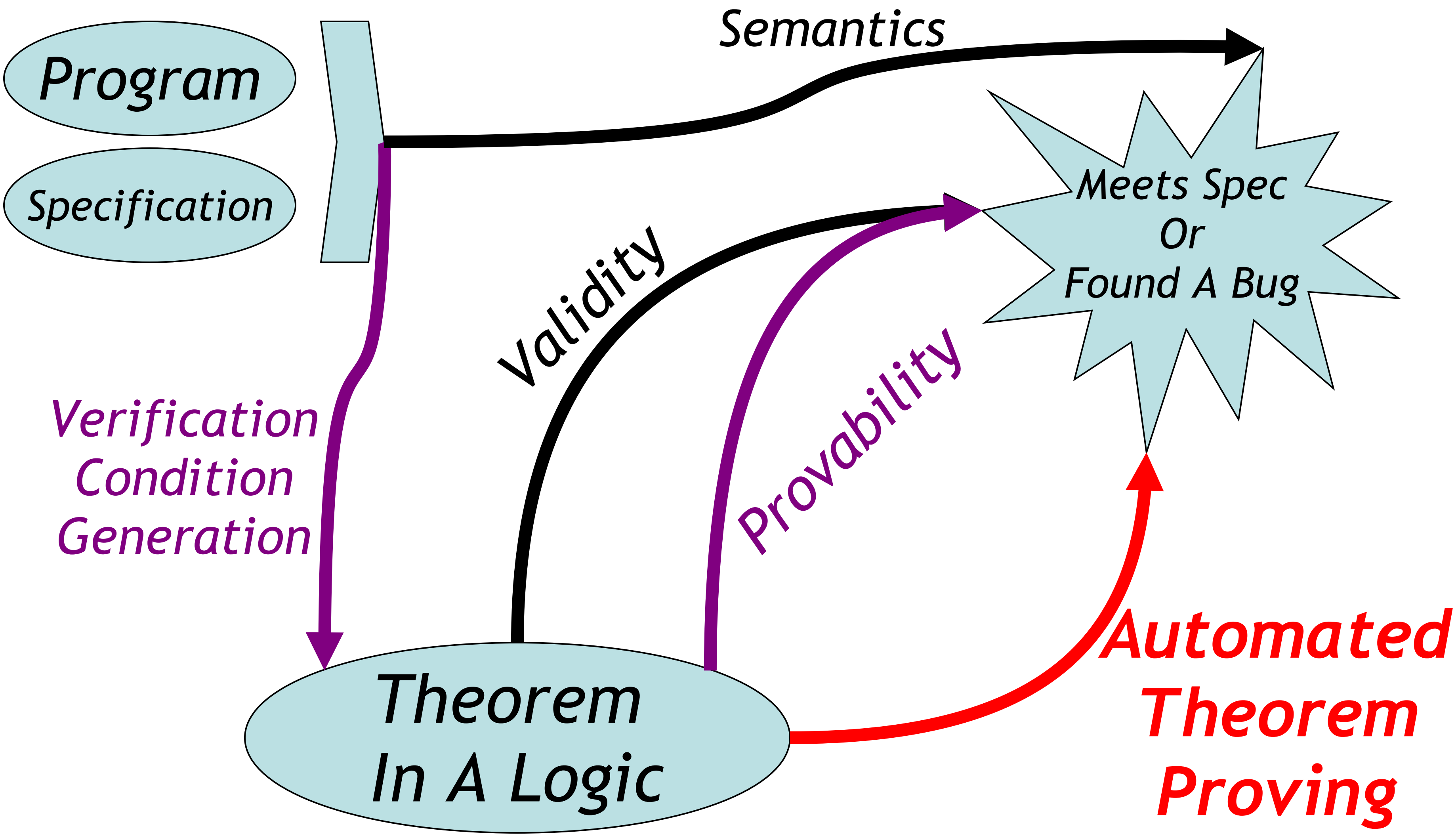
Motivation

- Can be viewed as "*decidable AI*"
  - Would be nice to have a procedure to automatically reason from premises to conclusions …
- Used to rule out the exploration of infeasible paths (model checking, dataflow)
- Used to reason about the heap (McCarthy, symbolic execution)
- Used to automatically synthesize programs from specifications (e.g. Leroy, Engler optional papers)
- Used to discover proofs of conjectures (e.g., Tarski conjecture proved by machine in 1996, efficient geometry theorem provers)
- Generally under-utilized

History

- Automated deduction is *logical deduction performed by a machine*
- Involves logic and mathematics
- One of the oldest and technically deepest fields of computer science
  - Some results are as much as 75 years old
  - "Checking a Large Routine", Turing 1949
  - Automation efforts are about 40 years old
  - Floyd-Hoare axiomatic semantics
- Still experimental (even after 40 years)

Standard Architecture



Logic Grammar

- We'll use the following logic:

Goals: $\mathbf{G ::= L \mid true \mid G_1 \wedge G_2 \mid H \Rightarrow G \mid \forall x.\ G}$

Hypotheses: $\mathbf{H ::= L \mid true \mid H_1 \wedge H_2}$

Literals: $\mathbf{L ::= p(E_1, \ldots, E_k)}$

Expressions: $\mathbf{E ::= n \mid f(E_1, \ldots, E_m)}$

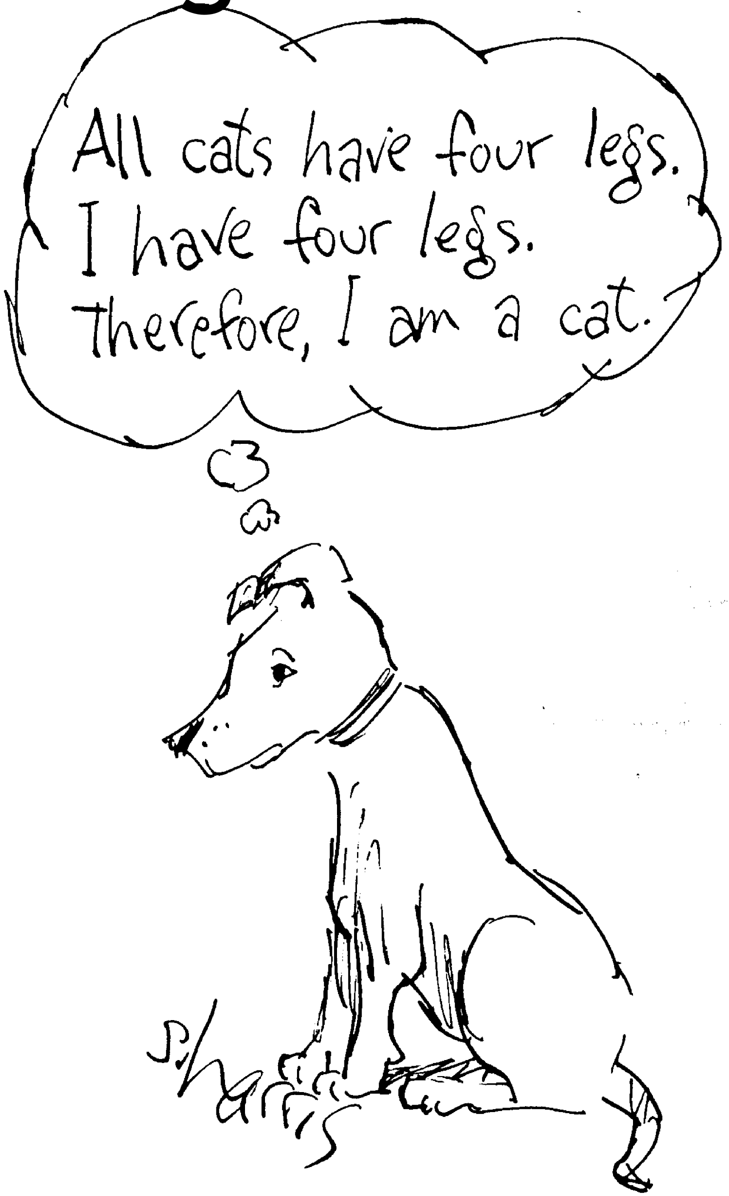
- This is a subset of first-order logic
  - Intentionally restricted: no $\vee$ so far
  - Predicate functions p: <, =, …
  - Expression functions f: +, *, sel, upd,

Theorem Proving Problem

- Write an algorithm "prove" such that:
- If prove(G) = true then $\models G$
  - Soundness (must have)
- If $\models G$ then prove(G) = true
  - Completeness (nice to have, optional)
- prove(H,G) means prove $H \Rightarrow G$
- Architecture: Separation of Concerns
  - #1. Handle $\wedge$, $\Rightarrow$, $\forall$, =
  - #2. Handle $\leq$, *, sel, upd, =

Theorem Proving

- Want to prove true things
- Avoid proving false things
- We'll do **proof-checking** later to rule out the "cat proof" shown here
- For now, let's just get to the point where we can prove something



Basic Symbolic Theorem Prover

- Let's define **prove(H,G)** …

prove(H, true) = true

prove(H, $G_1 \wedge G_2$) = prove(H,G_1) &&

prove(H, G_2)

prove(H_1, $H_2 \Rightarrow G$) = prove($H_1 \wedge H_2$, G)

prove(H, $\forall$x. G) = prove(H, G[a/x])

(a is "fresh")

prove(H, L) = ???

Theorem Prover for Literals

- We have reduced the problem to

  prove(H,L)

- But H is a conjunction of literals $L_1 \wedge \ldots \wedge L_k$
- Thus we really have to prove that

  $$L_1 \wedge \ldots \wedge L_k \Rightarrow L$$

- Equivalently, that $L_1 \wedge \ldots \wedge L_k \wedge \neg L$ is unsatisfiable
  - For any assignment of values to variables the truth value of the conjunction is false
- Now we can say

  prove(H,L) = Unsat(H $\wedge \neg$ L)

Theory Terminology

- A <u>theory</u> consists of a set of functions and predicate symbols (*syntax*) and definitions for the meanings of those symbols (*semantics*)
- Examples:
  - **0, 1, -1, 2, -3, …, +, -, =, <** (usual meanings; "theory of integers with arithmetic" or "Presburger arithmetic")
  - **=, $\leq$** (axioms of transitivity, anti-symmetry, and $\forall x.\ \forall y.\ x \leq y \vee y \leq x$; "theory of total orders")
  - **sel, upd** (McCarthy's "theory of lists")

Decision Procedures for Theories

- The Decision Problem
  - Decide whether a formula in a theory with first-order logic is true
- Example:
  - Decide "$\forall x.\ x>0 \Rightarrow (\exists y.\ x=y+1)$" in $\{\mathbb{N}, +, =, >\}$
- A theory is decidable when there is an algorithm that solves the decision problem
  - This algorithm is the decision procedure for that theory

Satisfiability Procedures

- The Satisfiability Problem
  - Decide whether a *conjunction of literals* in the theory is satisfiable
  - Factors out the first-order logic part
  - The decision problem can be reduced to the satisfiability problem
    - Parameters for ∀, skolem functions for ∃, negate and convert to DNF (sorry; I won't explain this here)
- "Easiest" Theory = Propositional Logic = SAT
  - A decision procedure for it is a "SAT solver"

Theory of Equality

- Theory of equality with *uninterpreted functions*
- Symbols: **=, ≠, f, g, …**
- Axiomatically defined (A,B,C ∈ Expressions):

$$\frac{}{A=A} \qquad \frac{B=A}{A=B} \qquad \frac{A=B \quad B=C}{A=C} \qquad \frac{A=B}{f(A)=f(B)}$$

- Example satisfiability problem:

$$g(g(g(x)))=x \wedge g(g(g(g(g(x)))))=x \wedge g(x)\neq x$$

More Satisfying Examples

- Theory of Linear Arithmetic
  - Symbols: $\geq$, **=**, **+**, **-**, **integers**
  - Example: $y > 2x + 1, x > 1, y < 0$ is unsat
  - Satisfiability problem is in P (loosely, no multiplication means no tricky encodings)
- Theory of Lists
  - Symbols: **cons, head, tail, nil**

$$\frac{}{\text{head}(\text{cons}(A,B)) = A} \qquad \frac{}{\text{tail}(\text{cons}(A,B) = B}$$

  - Theorem: $\text{head}(x) = \text{head}(y) \wedge \text{tail}(x) = \text{tail}(y) \Rightarrow x = y$

Computer Science

- This algorithmic strategy is applicable to decomposable problems that exhibit the optimal substructure property (in which the optimal solution to a problem P can be constructed from the optional solutions to its overlapping subproblems). The term was coined in the 1940's by Richard Bellman. Problems as diverse as "shortest path", "sequence alignment" and "CFG parsing" use this approach.

Mixed Theories

- Often we have facts involving *symbols from multiple theories*
  - E's symbols =, ≠, f, g, … (uninterp function equality)
  - R's symbols =, ≠, +, -, ≤, 0, 1, … (linear arithmetic)
  - Running Example (and Fact):
    $\models x \leq y \;\wedge\; y + z \leq x \;\wedge\; 0 \leq z \;\Rightarrow\; f(f(x) - f(y)) = f(z)$
  - To prove this, we must decide:
    Unsat$(x \leq y,\ y + z \leq x,\ 0 \leq z,\ f(f(x) - f(y)) \neq f(z))$
- We may have a sat procedure for each theory
  - E's sat procedure by Ackermann in 1924
  - R's proc by Fourier
- The sat proc for their combination is much harder
  - Only in 1979 did we get E+R

Satisfiability of Mixed Theories

$\text{Unsat}(x \le y,\ y + z \le x,\ 0 \le z,\ f(f(x) - f(y)) \ne f(z))$

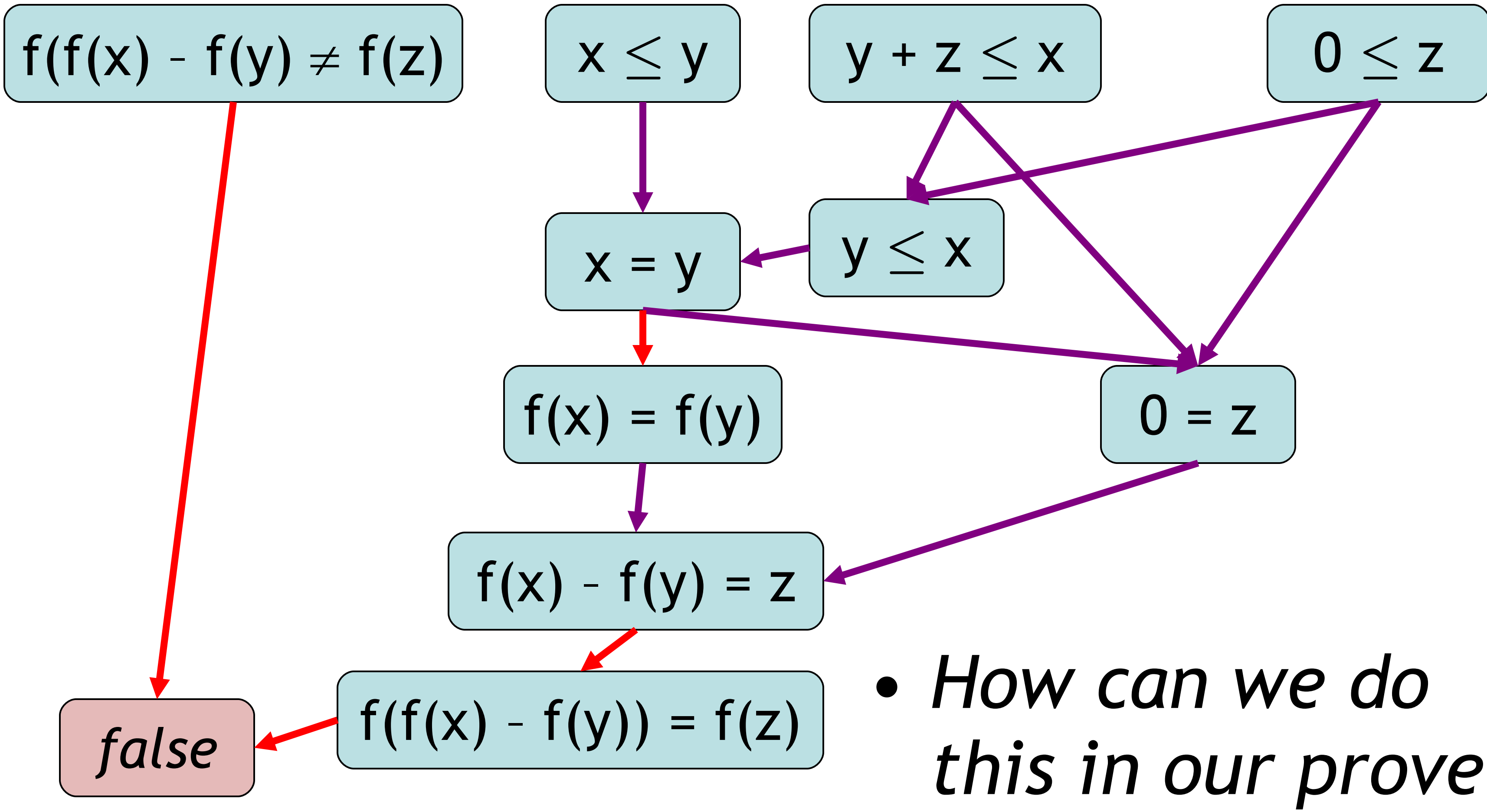
- Can we just separate out the terms in Theory 1 from the terms in Theory 2 and see if they are separately satisfiable?
  - No, unsound, equi-sat $\ne$ equivalent.
- The problem is that the two satisfying assignments may be incompatible
- Idea (Nelson and Oppen): Each sat proc announces all equalities between variables that it discovers

Handling Multiple Theories

- We'll use cooperating decision procedures
- Each sat proc works on the literals it understands
- Sat procs share information (equalities)

"THEN, AS YOU CAN SEE, WE GIVE THEM SOME MULTIPLE CHOICE TESTS."

Consider Equality and Arith

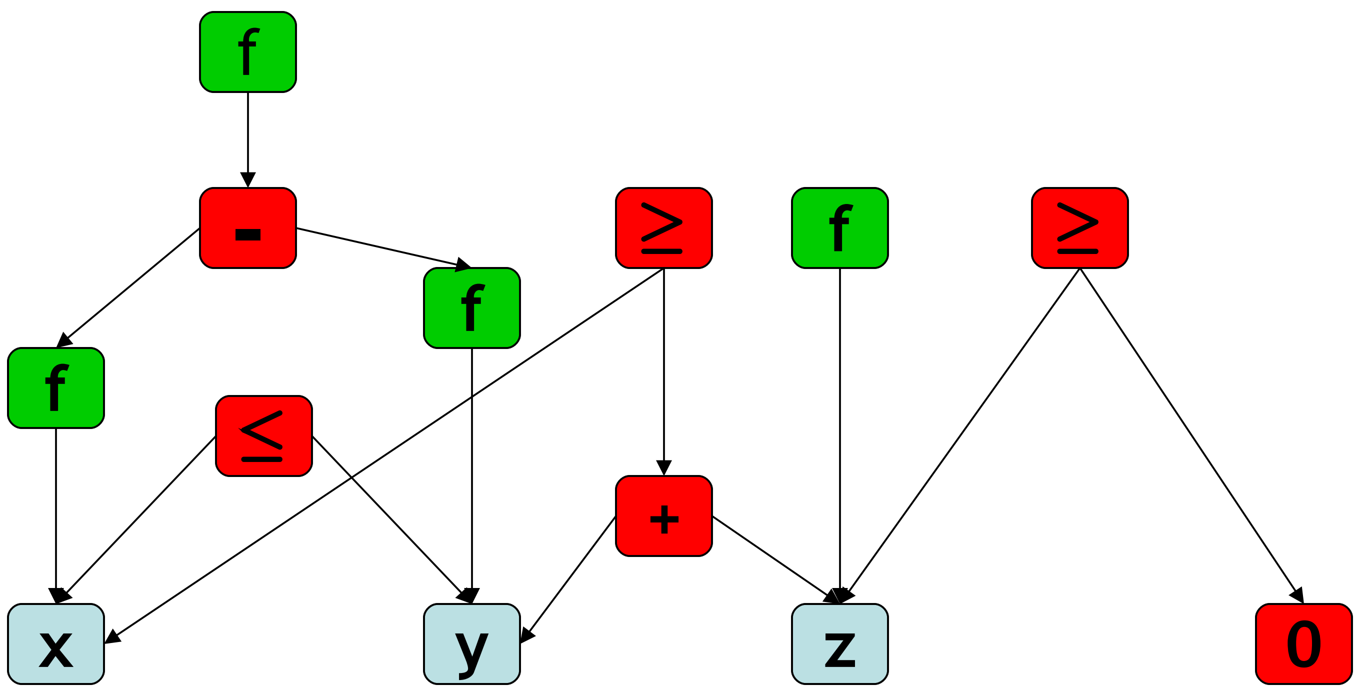


- *How can we do this in our prover?*

Nelson-Oppen: The E-DAG

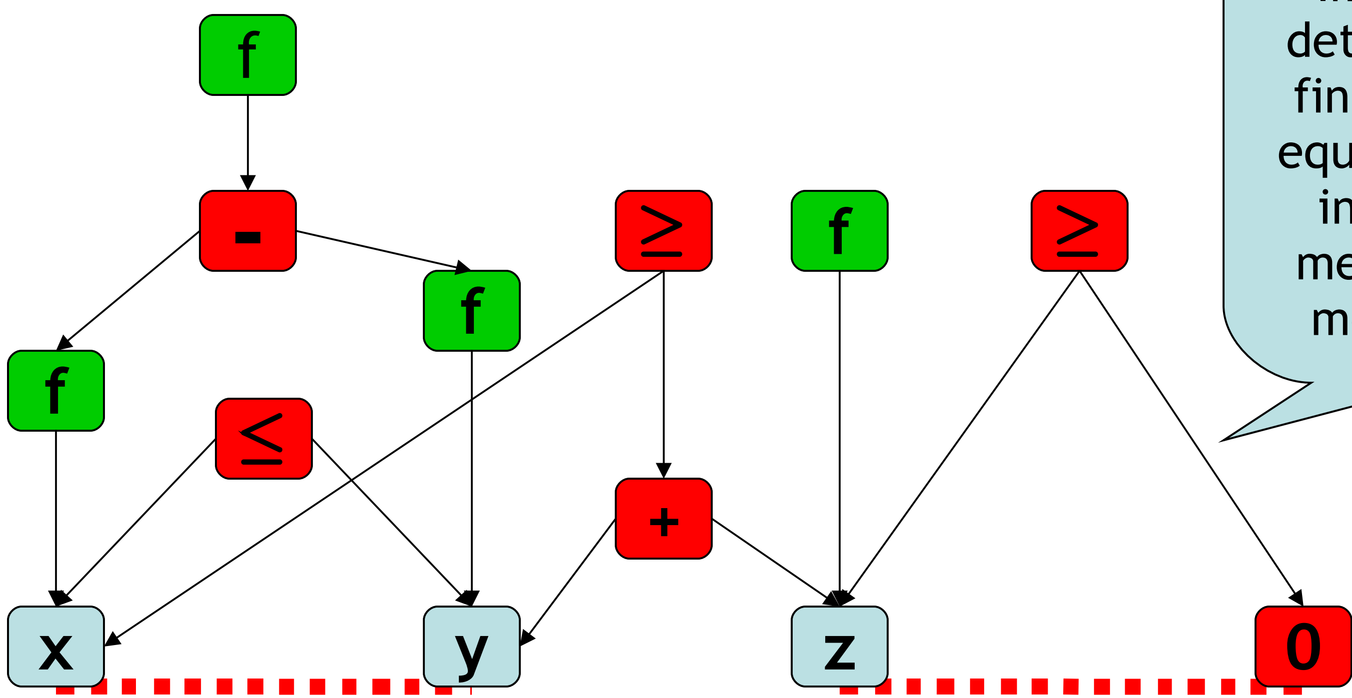
- Represent all terms in one Equivalence DAG
  - Node names act as variables shared between theories!

$$f(f(x) - f(y)) \neq f(z) \wedge y \geq x \wedge x \geq y + z \wedge z \geq 0$$



Nelson-Oppen: Processing

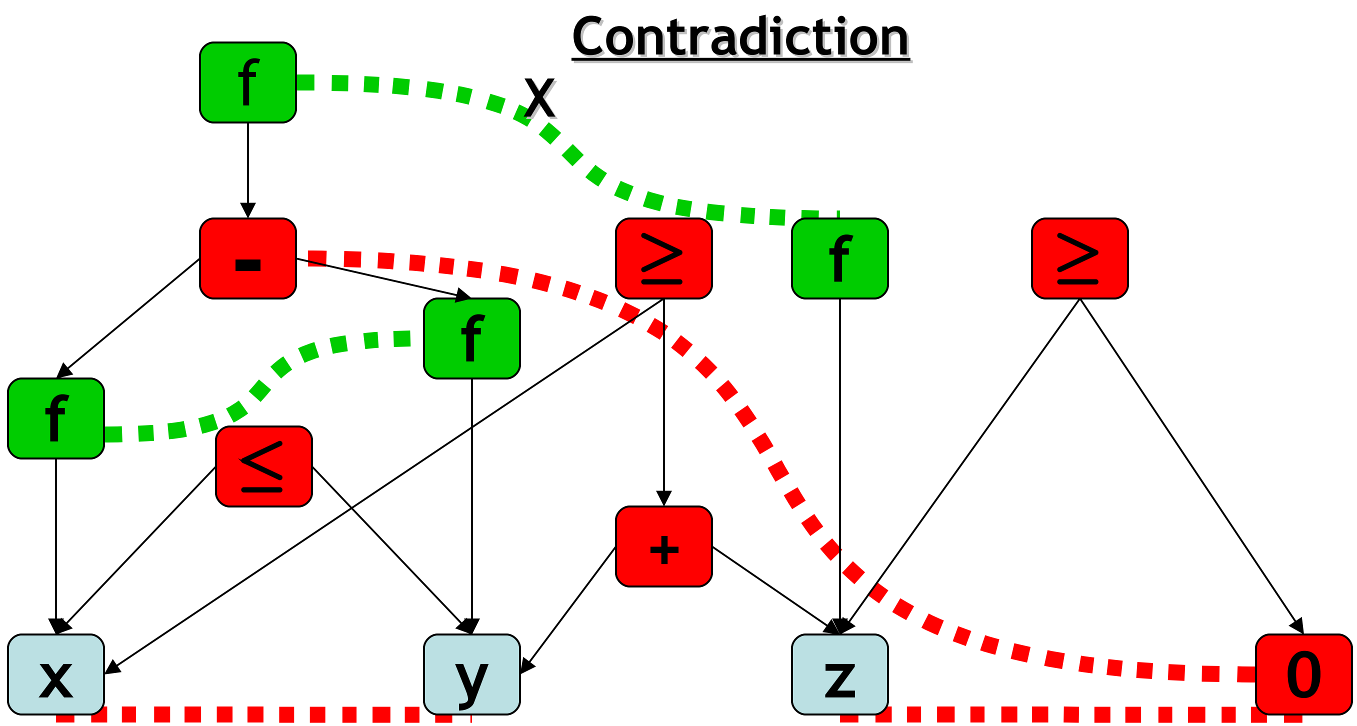
- Run each sat proc
  - Report all contradictions (as usual)
  - Report all equalities between nodes (key idea)



Implementation details: Use union-find to track node equivalence classes in E-DAG. When merging A=B, also merge f(A)=f(B).

Nelson-Oppen: Processing

- Broadcast all discovered equalities
  - Rerun sat procedures
  - Until no more equalities or a contradiction



Does It Work?

- If a contradiction is found, then unsat
  - This is sound if sat procs are sound
  - Because only sound equalities are ever found
- If there are no more equalities, then sat
  - Is this complete? Have they shared enough info?
  - Are the two satisfying assignments compatible?
  - Yes!
  - *(Countable theories with infinite models admit isomorphic models, convex theories have necessary interpretations, etc.)*

SAT-Based Theorem Provers

- Recall separation of concerns:
  - #1 Prover handles connectives ($\forall$, $\wedge$, $\Rightarrow$)
  - #2 Sat procs handle literals (+, $\leq$, 0, head)
- Idea: reduce proof obligation into propositional logic, feed to SAT solver (CVC)
  - To Prove: $3*x=9 \Rightarrow (x = 7 \wedge x \leq 4)$
  - Becomes Prove: $A \Rightarrow (B \wedge C)$
  - Becomes Unsat: $A \wedge \neg(B \wedge C)$
  - Becomes Unsat: $A \wedge (\neg B \vee \neg C)$

SAT-Based Theorem Proving

- To Prove: $3*x=9 \Rightarrow (x = 7 \wedge x \leq 4)$
  - Becomes Unsat: $A \wedge (\neg B \vee \neg C)$
  - SAT Solver Returns: A=1, C=0
  - Ask sat proc: unsat($3*x=9$, $\neg$ $x \leq 4$) = true
  - Add constraint: $\neg(A \wedge \neg C)$
  - Becomes Unsat: $A \wedge (\neg B \vee \neg C) \wedge \neg(A \wedge \neg C)$
  - SAT Solver Returns: A=1, B=0, C=1
  - Ask sat proc: unsat($3*x=9$, $\neg$ $x=7$, $x \leq 4$) = false
    - ($x=3$ is a satisfying assignment)
  - We're done! (original to-prove goal is false)
  - If SAT Solver returns "no satisfying assignment" then original to-prove goal is true

Proofs

"Checking proofs ain't like dustin' crops, boy!"

Proof Generation

- We want our theorem prover to **emit proofs**
  - No need to trust the prover
  - Can find bugs in the prover
  - Can be used for proof-carrying code
  - Can be used to extract invariants
  - Can be used to extract models (e.g., in SLAM)
- Implements the soundness argument
  - On every run, a soundness proof is constructed

Proof Representation

- Proofs are trees
  - Leaves are hypotheses/axioms
  - Internal nodes are inference rules
- Axiom: “true introduction”
  - Constant: truei : pf
  - pf is the type of proofs
- Inference: “conjunction introduction”
  - Constant: andi : pf $\rightarrow$ pf $\rightarrow$ pf
- Inference: “conjunction elimination”
  - Constant: andel : pf $\rightarrow$ Pf
- Problem:
  - “andel truei : pf” but does not represent a valid proof
  - Need a more powerful *type system that checks content*

$$\frac{}{\vdash \text{true}}\ \textit{truei}$$

$$\frac{\vdash A \qquad \vdash B}{\vdash A \wedge B}\ \textit{andi}$$

$$\frac{\vdash A \wedge B}{\vdash A}\ \textit{andel}$$

Dependent Types

- Make pf a family of types indexed by formulas
  - f : Type (type of encodings of formulas)
  - e : Type (type of encodings of expressions)
  - pf : f $\rightarrow$ Type (the type of proofs indexed by formulas: it is a proof ***that f is true***)
- Examples:
  - true : f
  - and : f $\rightarrow$ f $\rightarrow$ f
  - truei : pf true
  - andi : pf A $\rightarrow$ pf B $\rightarrow$ pf (and A B)
  - **andi : ΠA:f. ΠB:f. pf A $\rightarrow$ pf B $\rightarrow$ pf (and A B)**
  - **(ΠA:f.X means "forall A of type f, dependent type X", see next lecture)**

Proof Checking

- Validate proof trees by type-checking them
- Given a proof tree X claiming to prove A ∧ B
- Must check X : pf (and A B)
- We use "expression tree equality", so
  - andel (andi "1+2=3" "x=y") does *not* have type pf (3=3)
  - This is already a proof system! If the proof-supplier wants to use the fact that 1+2=3 ⇔ 3=3, she can include a proof of it somewhere!
- Thus Type Checking = Proof Checking
  - And it's quite easily *decidable*! □

Parametric Judgment (Time?)

- Universal Introduction Rule of Inference

$$\frac{\vdash [a/x]A \quad \text{(a is fresh)}}{\vdash \forall x.\ A}$$

- We represent bound variables in the logic using bound variables in the meta-logic
  - all : $(e \rightarrow f) \rightarrow f$
  - Example: $\forall x.\ x=x$ represented as (all (λx. eq x x))
  - Note: $\forall y.\ y=y$ has an α-equivalent representation
  - Substitution is done by β-reduction in meta-logic
    - **[E/x](x=x) is** **(λx. eq x x) E**

Parametric ∀ Proof Rules (Time?)

$$\frac{\vdash [a/x]A \quad \text{(a is fresh)}}{\vdash \forall x.\ A}$$

- Universal Introduction
  - alli: $\Pi A{:}(e \to f).\ (\Pi a{:}e.\ \text{pf}\ (A\ a)) \to \text{pf}\ (\text{all}\ A)$

$$\frac{\vdash \forall x.\ A}{\vdash [E/x]A}$$

- Universal Elimination
  - alle: $\Pi A{:}(e \to f).\ \Pi E{:}e.\ \text{pf}\ (\text{all}\ A) \to \text{pf}\ (A\ E)$

Parametric ∃ Proof Rules (Time?)

$$\frac{\vdash [E/x]A}{\vdash \exists x.\ A}$$

- Existential Introduction
  - existi: ΠA:(e $\rightarrow$ f). ΠE:e. pf (A E) $\rightarrow$ pf (exists A)

$$\frac{\vdash \exists x.\ A \qquad \begin{array}{c}\vdash [a/x]A \\ \dots \\ \vdash B\end{array}}{\vdash B}$$

- Existential Elimination
  - existe: ΠA:(e $\rightarrow$ f). ΠB:f.

    pf (exists A) $\rightarrow$ (Πa:e. pf (A a) $\rightarrow$ pf B) $\rightarrow$ pf B

Homework

- Project
  - Need help? Stop by my office or send email.