Equality

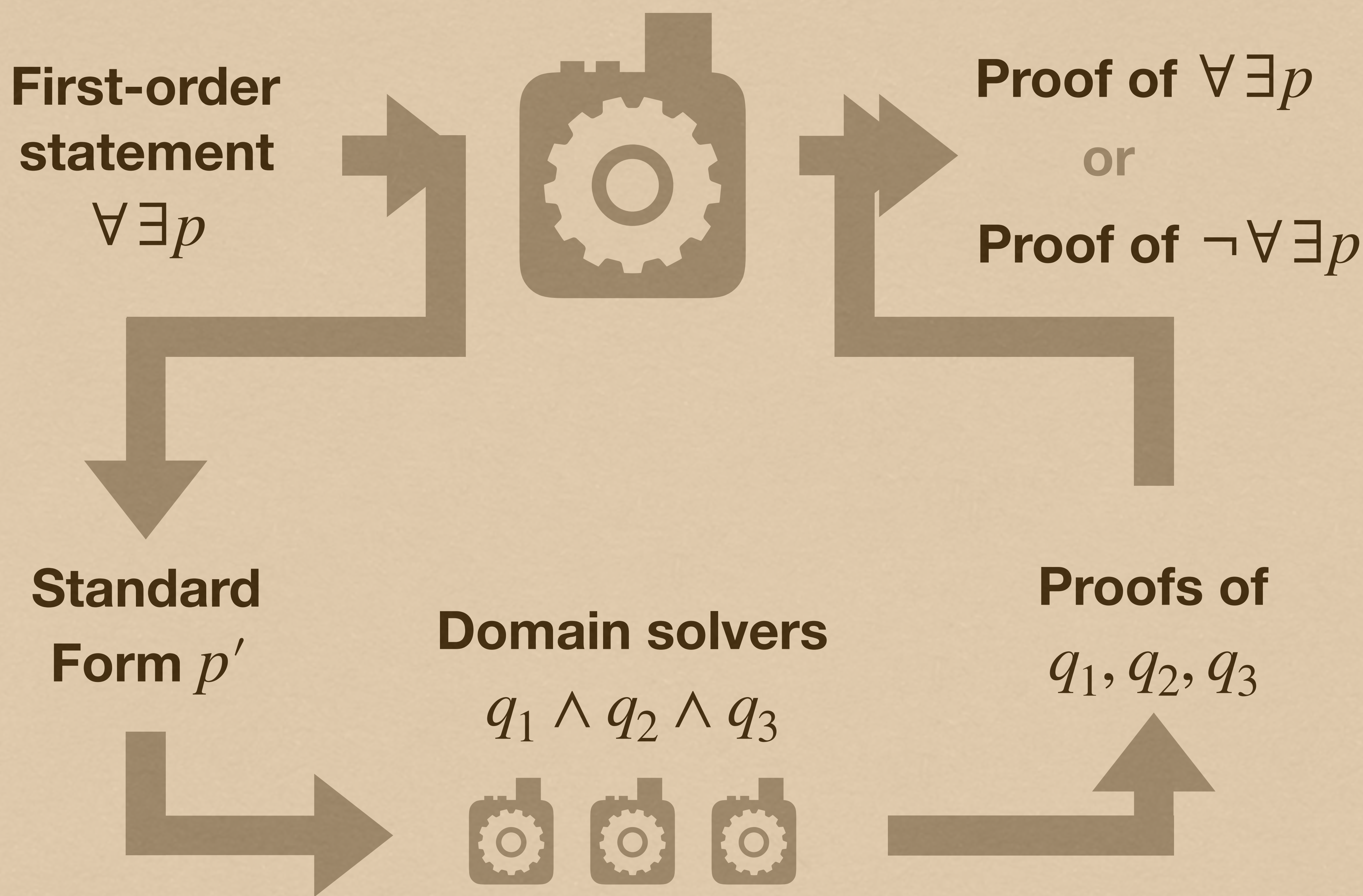
Specifications section, **Algorithms** topic, **Lecture 7**

Pavel Panchekha

CS 6110, U of Utah

28 January 2020

The Approach



DPLL with Theories

Want **extension to first-order theories**

Statement p

$$(\neg a_1 \vee a_2) \wedge (\neg b_1 \vee b_2) \wedge \ldots$$

Assignment Γ

$$a_1 \wedge \neg b_2 \wedge \ldots$$

The $a_i, b_i, \ldots$ are **literals from the theory**

$$(\neg(\underbrace{x < 0}_{a_1}) \vee (\underbrace{x \times x > 0}_{a_2})) \wedge (\neg(\underbrace{x = 0}_{b_1}) \vee (\underbrace{x \times x = 0}_{b_2}) \wedge \ldots$$

$$\underbrace{x < 0}_{a_1} \wedge (\underbrace{x \times x = 0}_{b_2}) \wedge \ldots$$

Standard Form

$\neg\ G$

$\wedge\ E = a[C := B]$

$\wedge\ F = E[D]$

$\wedge\ G = F < A$

$\wedge\ D = y + x$

$\wedge\ C = x + y$

$\wedge\ B = 5$

$\wedge\ A = 6$

Choose C = D

SAT

a	B	C	D	E	F
?1 → ?2 ?3 → ?4	?5	?1	?3	?1 → ?5 ?3 → ?4	?4

C = D? Theories disagree

SAT

x	y	A	B	C	D	F
0	0	6	5	0	0	3

Class Progress

Logical reasoning

Program logics

Static analysis

First-order Logic

Decision Procedures

Mixing Theories

Equality

Integers

Arrays

Equality

Turning syntactic equality into **pointer equality**

Pointer sharing in tree structures

Solving equality queries with congruence closure

Rebuilding to ensure accuracy

Model-based quantifier instantiation

And why the heuristic doesn’t work in general

Term Databases

A model of the reflection axioms

The Solver Query

Statement p

$$(\neg a_1 \vee a_2) \wedge (\neg b_1 \vee b_2) \wedge \ldots$$

Assignment Γ

$$a_1 \wedge \neg b_2 \wedge \ldots$$

Today: a solver for **equality queries**

$$\Gamma = (s_1 = t_1) \wedge (s_2 = t_2) \wedge (s_3 \neq t_3) \wedge \ldots$$

$$\Gamma = a = b \wedge b = c \wedge a \neq c$$

$$\Gamma = a = b \wedge f(a) \neq f(b)$$

$$\Gamma = f(a) = b \wedge f(b) = a \wedge a \neq b$$

Modeling

Goal is to check all relevant instances of **axioms**

Reflection

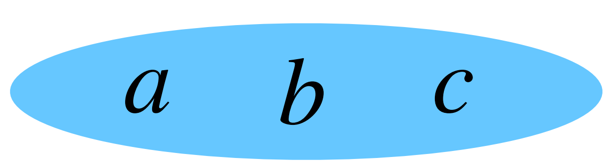
$$\forall x, x = x$$

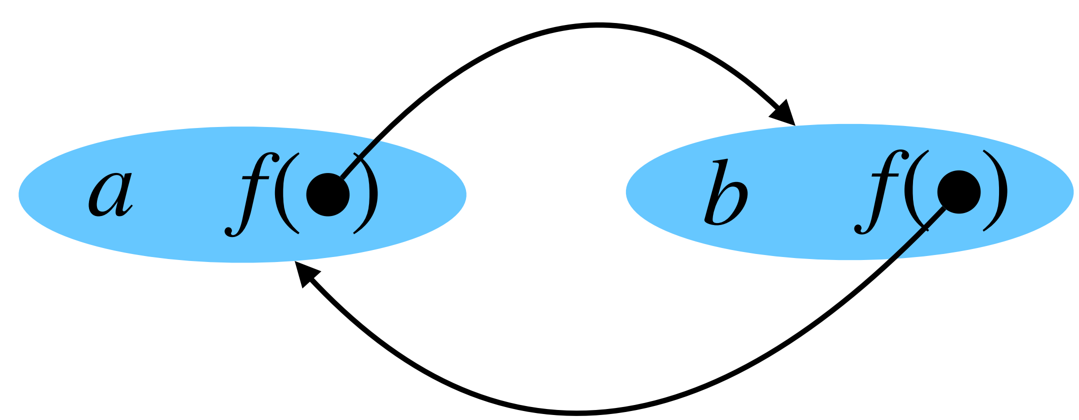
Substitution

$$\forall x, \forall y, x = y \rightarrow f(x) = f(y)$$

Intuitive approach **starts with the true** equalities

Build **model** of assumptions, check conclusions





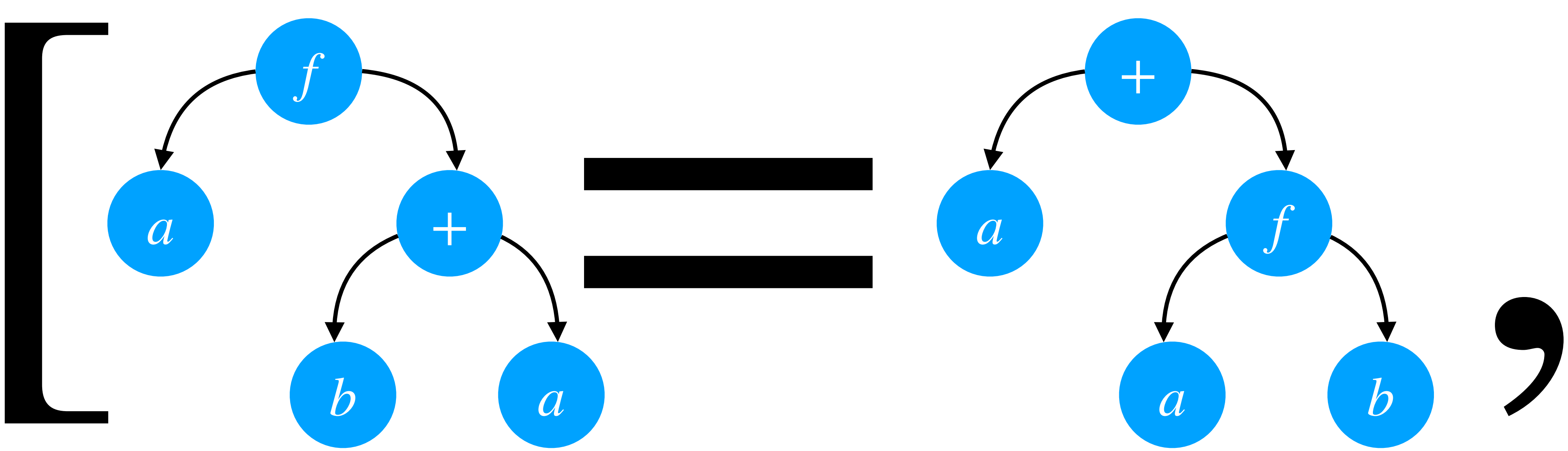
Term Representation

Query is a **list of pairs of terms**, plus a flag for negation

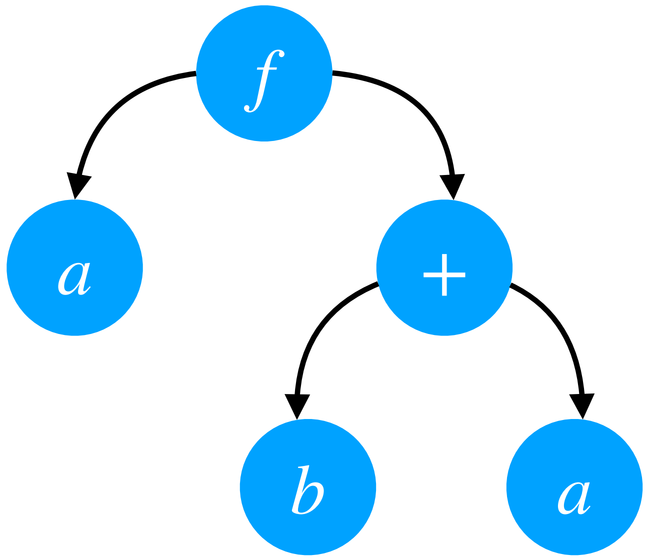
$$\Gamma = (s_1 = t_1) \wedge (s_2 = t_2) \wedge (s_3 \neq t_3) \wedge \ldots$$

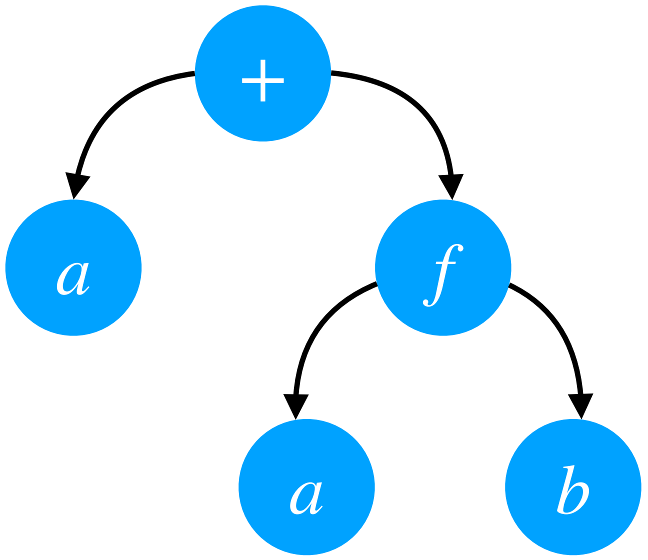
```
type EqQuery = List (Boolean, Term, Term)
```

Terms are **syntax trees**:

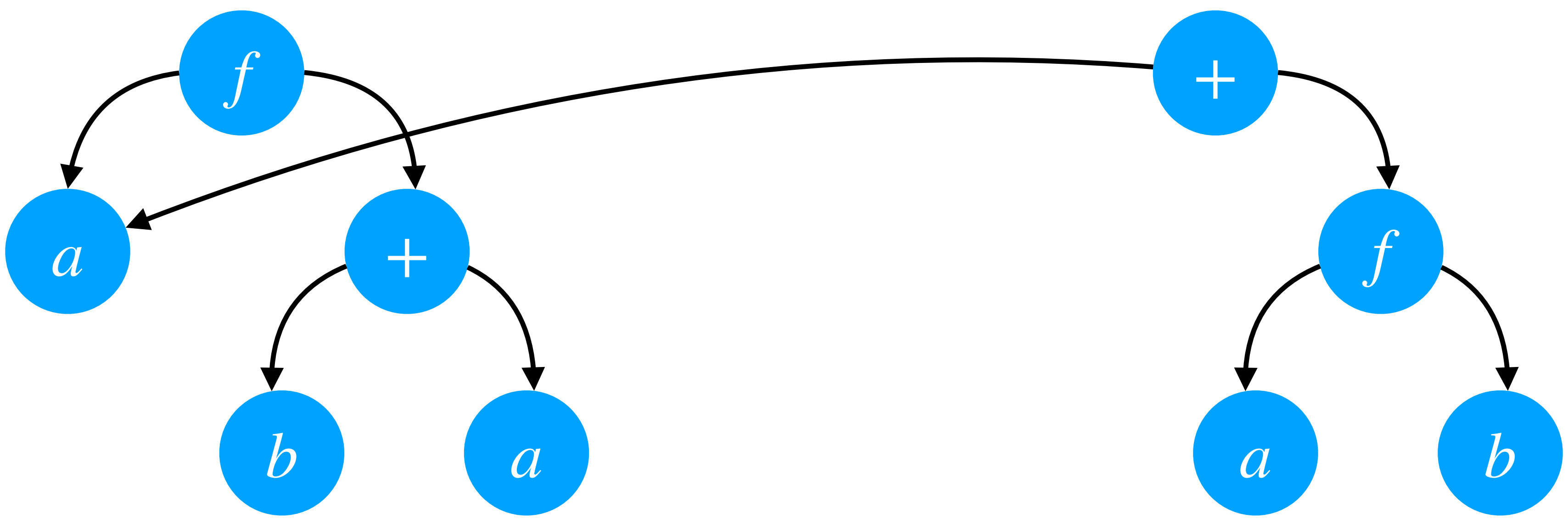


Subterm Sharing





Subterm Sharing



Avoid **multiple copies** of one expression

Make equal things identical to enforce equality axioms

Term Hashing

Create **term database** by assigning names to terms

```
class TermDB:

  def add(this : TermDB, term : Term) → Int
```

`add` recursively **reuses names** when possible

```
def add(this, term):
  arg_names = map(this.add, term.arguments)
  key = (term.function, arg_names)
```

Term Database

Equality solver **uses term database**

```
def equality_solver(query : EqQuery) → Bool:
  db = TermDb()
```

Term database **models reflection axiom**

- Same expression **has identical name** in database
- Need ability to **add new equalities**

Example

How many terms in the term database?

$$f(g(a,1),2) = 5 \wedge g(f(1,5), g(a,1)) = 3 \wedge g(a,1) \neq a$$

1	1		
2	2		
3	3		
4	5		
5	a		

6	g	5	1
7	f	1	4
8	g	7	6
9	f	6	2

Basic Checking

Need to **check satisfiability** after adding all terms

$$\Gamma = s_1 = t_1 \wedge s_2 = t_2 \wedge s_3 \neq t_3 \wedge s_4 \neq t_4$$

$$\neg\Gamma = (s_1 = t_1 \wedge s_2 = t_2) \rightarrow (s_3 = t_3 \vee s_4 = t_4)$$

Check disequalities via names

```
neqs = [(s, t) for (iseq, s, t) in query if not iseq]

return not any(db.add(s) == db.add(t) for (s, t) in neqs)
```

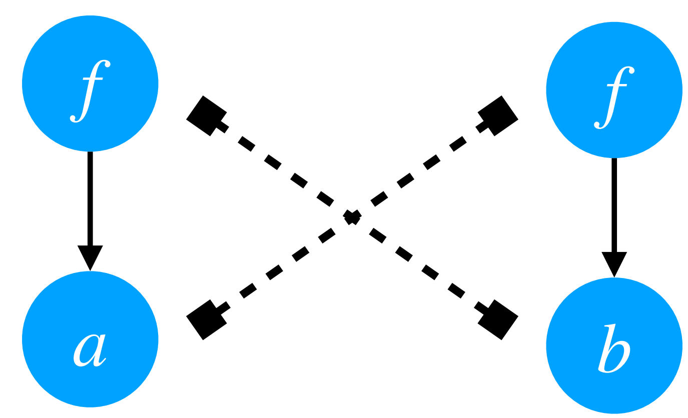
Equivalence Graphs

Adding assumptions to the term database

Adding equalities

$$\Gamma = f(a) = b \land f(b) = a \land a \neq b$$

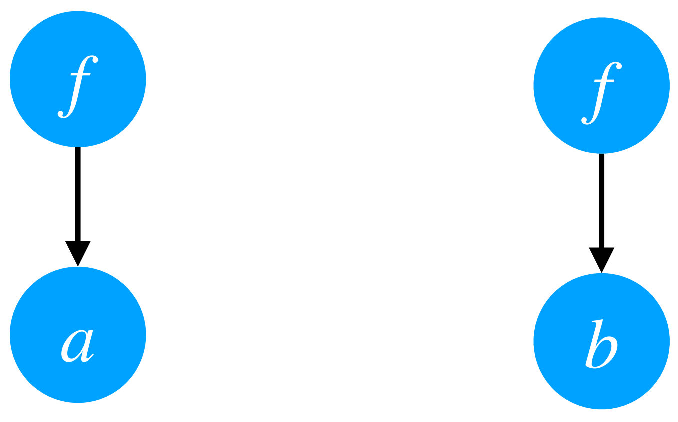
Need to **add equalities** to the term database



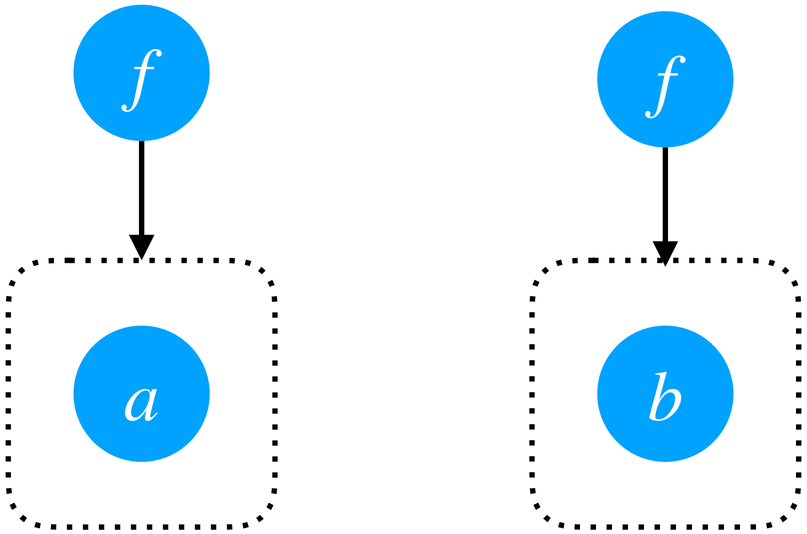
Must respect **substitution rule**:

$$a = b \rightarrow f(a) = f(b)$$

Equivalence Classes

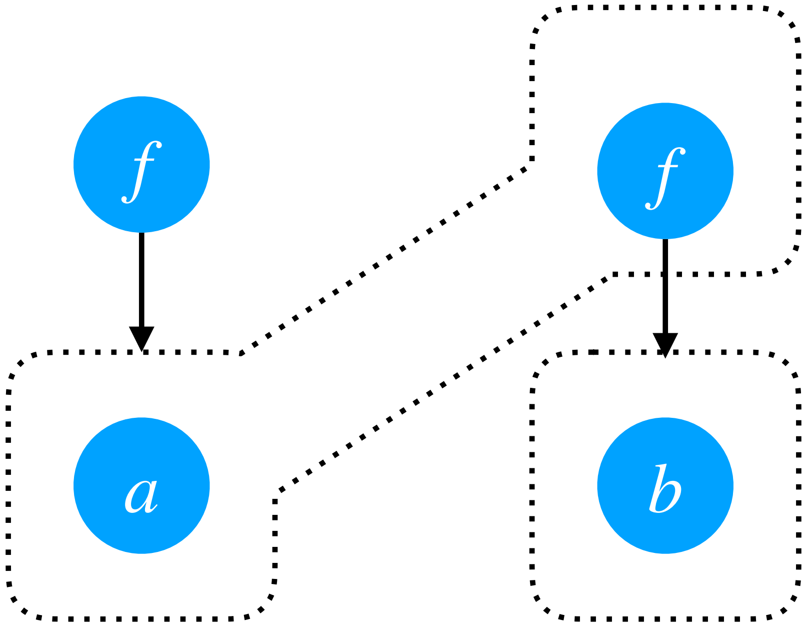


Equivalence Classes



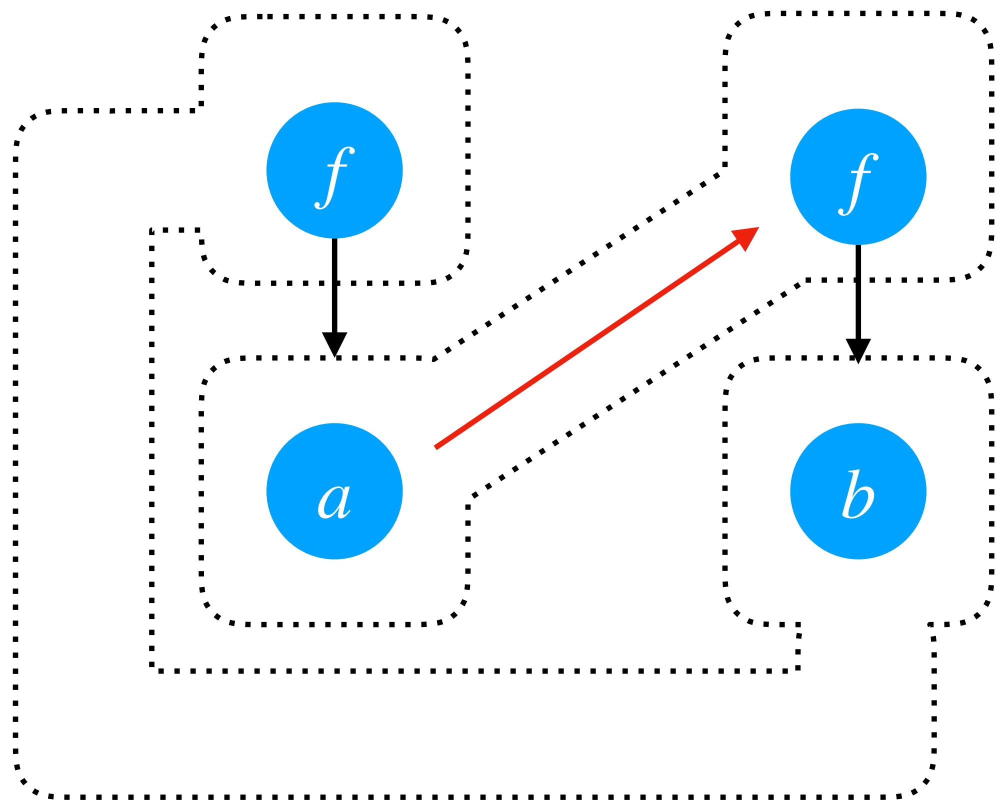
$$f(b) = a$$

Equivalence Classes



$$f(a) = b \qquad f(b) = a$$

Equivalence Classes



$$f(a) = b \qquad f(b) = a$$

How to represent equivalence classes **in code**?

Equivalence Classes

How to represent equivalence classes **in code**?

```
class TermDB:
  eqclass : Dict Int Int
                 Name Class
```

Initially, each node in **its own class**:

```
def add(this, term):
  # …
  this.eqclass[name] = name
```

Equivalence Classes

On merging, **pick one class** as the winner

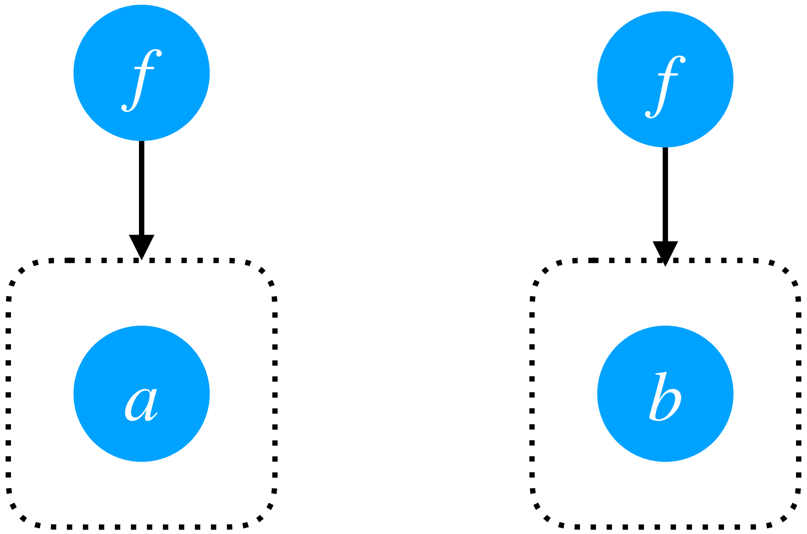
```
def merge(term1, term2):

  this.eqclass[class1] = class2
```

Class of a node determined **recursively**:

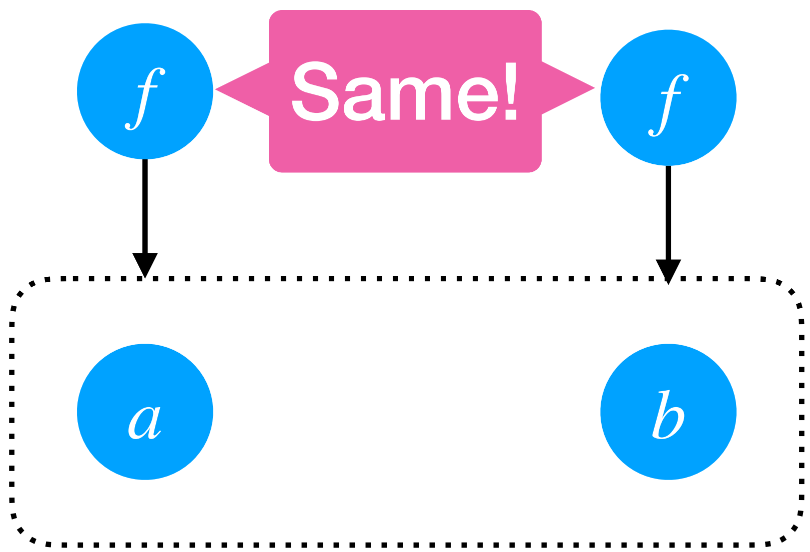
```
def classof(name):
  cls = this.eqclass[name]
  if cls == name: return pls
  else: return this.classof(cls)
```

Congruence Closure



$$a = b$$

Congruence Closure

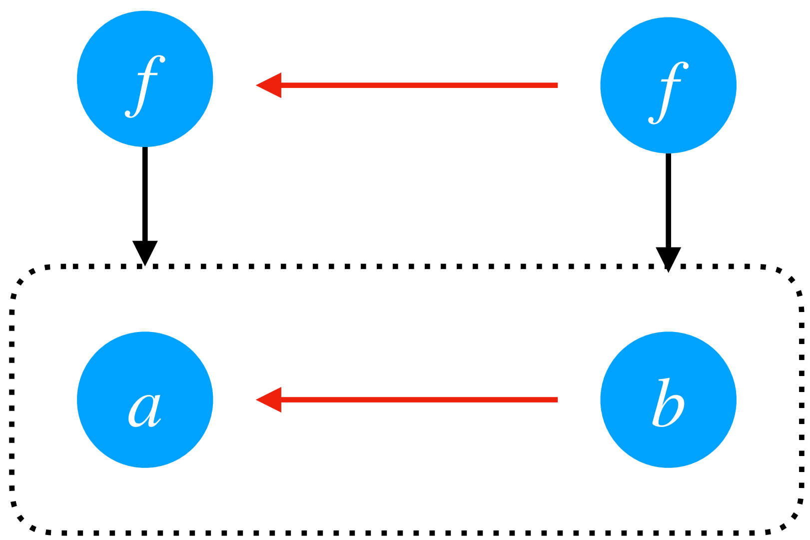


$$a = b$$

Merging two nodes can **make other nodes equal**

Need to **merge** those nodes to ensure correctness

Congruence Closure




```
for (f, args), name in this.names.items():
  arg_classes = map(this.classof, args)
  name2 = this.add(f, arg_classes)
  this.eqclass[name] = name2
```

Checking

Use “`merge`” on **true equalities**:

```
eqs = [(s, t) for (iseq, s, t) in query if iseq]

for s, t in eqs:
  this.merge(s, t)

this.merge_upward()
```

Check false equalities **at the end**

Example

How many classes in the term database?

$$f(a,b) = c \wedge f(b,c) = f(c,b) \wedge$$
$$f(f(a,b),b) = f(a,a) \wedge f(a,b) = f(a,a)$$

1		a		
2		b		
3		c		
4		f	1	2
5		f	2	3

6		f	3	2
7		f	3	2
9		f	1	1

Speed

Lots of ways to **make this faster**

- Fix up congruence closure **less often** (once at the end?)
- Pick which **class name to keep** when you merge
- **Cache** "`classof`" lookup so that it's fast
- **Lay out data structures** to be parallel, cache-friendly

Core data structure is as described

Course Updates

Assignment 2

Assignment 2

Assignment 2 **due on Thursday**

- **Submit early** so you're not late
- **Check submission** for any mistakes
- Please put your **name in file name**; it helps grading

Lots of good **questions on Piazza**

- → How to encode KenKen into SAT
- → How fast to expect the SMT solution to be
- → Hints on how the bonus problem should work

Quantifier Heuristics

Where quantifiers can and can’t be useful

Quantified Equalities

Quantifiers convenient to **express arithmetic laws**:

$$\forall x \forall y, x + y = y + x$$

$$\forall x \forall y, (x + y) + z = x + (y + z)$$

$$\forall x \forall y, (x + y) - y = x$$

Can **equivalence graphs use quantified** equalities?

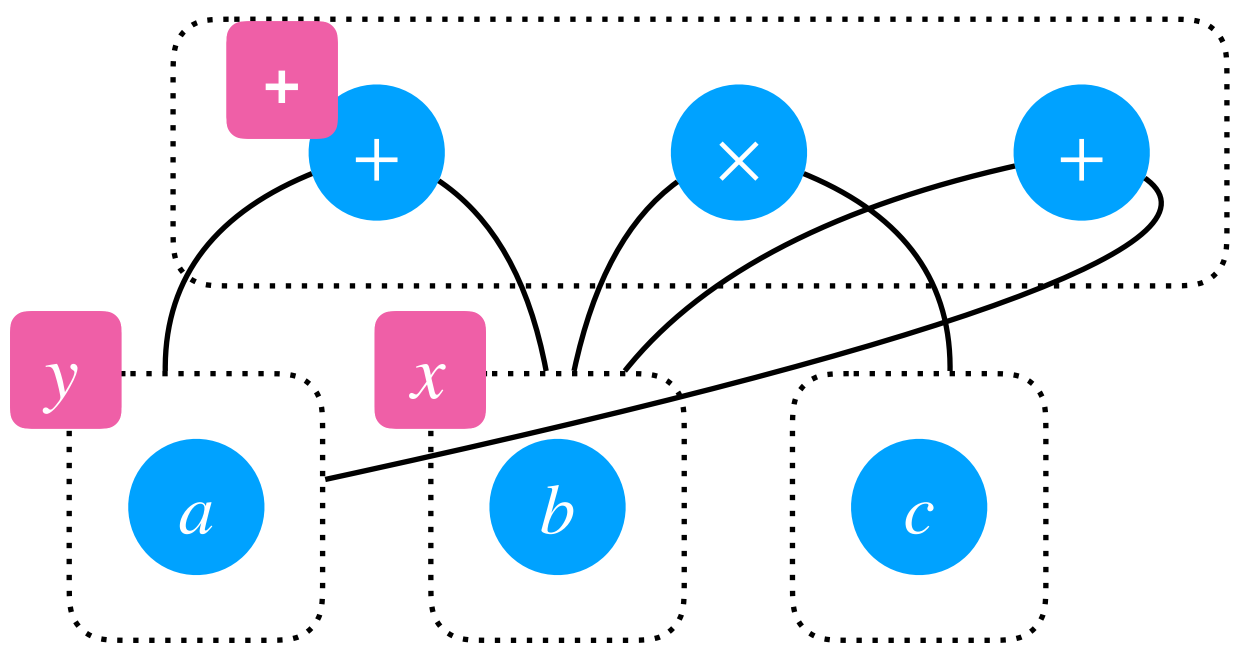
Instantiation

Quantified equality is **infinitely-many** equalities

But only some of them are **relevant**

Instantiate based on **content of e-graph** (model-based)

$$\forall x \forall y, x + y = y + x$$



Try both directions!

E-matching

Goal: find places a **quantified equality matches**

1. Loop over all e-classes
2. Find terms with **matching function name**
3. **Recurse** on arguments
4. Check variables **bound identically**

$$\forall x, x - x = 0$$

Bottom-up Matching

E-matching **wastes a lot of work**

$$\forall x, \forall y, \forall z, x + (y + z) = (x + y) + z$$

More efficient to match **bottom-up**

- **Combine** all patterns from quantified equalities
- Search for **leaf nodes first**, recording matching classes
- Higher-level patterns **refer to recorded matches**

RETE algorithm combines patterns into **state machine**

Complexity

$$((x + y) + (z + w)) + ((u + v) + (s + t))$$

Commutativity grows the term database… **linearly**

$$\forall x \forall y, x + y = y + x$$

Associativity grows the term database… **exponentially**

$$\forall x \forall y, (x + y) + z = x + (y + z)$$

Inversion grows the term database… **infinitely**

$$\forall x \forall y, f^{-1}(f(x)) = x$$

Next class:

Integers

To do:

- ☐ Course feedback
- ☐ Read Chapter 9
- ☐ Assignment 2

Equality

Turning syntactic equality into **pointer equality**

Pointer sharing in tree structures

Solving equality queries with congruence closure

Rebuilding to ensure accuracy

Model-based quantifier instantiation

And why the heuristic doesn't work in general

EQUALITIES

ELIMINATION

SUBSTITUTION

INEQUALITIES

ELIMINATING VARIABLES

SPEED

LINEAR

OPTIMIZATION

Next class:

Integers

To do:

- ☐ Course feedback
- ☐ Read Chapter 9
- ☐ Assignment 2