Lecture 16

Pete Manolios
Northeastern

FOL Checking with Unification

- FO validity checker: Given FO ϕ, negate & Skolemize to get universal ψ s.t. Valid(ϕ) iff UNSAT(ψ). Let G be the set of ground instances of ψ (possibly infinite, but countable). Let G_1, G_2 …, be a sequence of finite subsets of G s.t. $\forall g \subseteq G, |g| < \omega$, $\exists n$ s.t. $g \subseteq G_n$. $\exists n$ s.t. Unsat G_n iff Unsat ψ (and Valid ϕ)
- Unification: intelligently instantiate formulas
- FO validity checker w/ unification: Given FO ϕ, negate & Skolemize to get universal ψ s.t. Valid(ϕ) iff UNSAT(ψ). Convert ψ into equivalent CNF $\mathcal{K}$. Then, Unsat ψ iff $\emptyset \in \text{URes}_\omega(\mathcal{K})$ iff $\exists n$ s.t. $\emptyset \in \text{URes}_n(\mathcal{K})$.
- We say that U-resolution is *refutation-compete*: If Unsat($\mathcal{K}$) then there is a proof using U-resolution (*i.e.*, you can derive $\emptyset$), so we have a semi-decision procedure for validity.

FOL Checking Examples

- FO validity checker w/ unification: Given FO φ, negate & Skolemize to get universal ψ s.t. Valid(φ) iff UNSAT(ψ). Convert ψ into equivalent CNF $\mathcal{K}$. Then, Unsat(ψ) iff $\emptyset \in \mathrm{URes}_{\omega}(\mathcal{K})$ iff ∃*n* s.t. $\emptyset \in \mathrm{URes}_{n}(\mathcal{K})$.

$$\phi = \neg\langle \forall x, y\ (R(x,y) \vee Q(x)) \ \wedge\ \neg R(x, g(x)) \ \wedge\ \neg Q(y)\rangle$$

$$\psi = \langle \forall x, y\ (R(x,y) \vee Q(x)) \ \wedge\ \neg R(x, g(x)) \ \wedge\ \neg Q(y)\rangle$$

$$\mathcal{K} = \{\{R(x,y), Q(x)\}, \{\neg R(x, g(x))\}, \{\neg Q(y)\}\}$$

$$\begin{tikzcd}
\{(R(x,y), \underline{Q(x)}\} \arrow[dr, dash] & & \{\underline{\neg Q(z)}\} \arrow[dl, dash, "\sigma = z \leftarrow x"] & \\
& \{(\underline{R(x,y)}\} \arrow[dr, dash, "\sigma = z \leftarrow x, y \leftarrow g(x)"'] & & \{\underline{\neg R(z, g(z))}\} \arrow[dl, dash] \\
& & \emptyset &
\end{tikzcd}$$

Let *C*, *D* be clauses (w/ no common variables). *K* is a U-resolvent of *C*, *D* iff there are non-empty $\underline{C}' \subseteq C$, $\underline{D}' \subseteq D$ s.t. σ is a unifier for $\underline{C}' \cup \underline{D}'^{-}$ and $K = (C \setminus \underline{C}' \cup D \setminus \underline{D}')\sigma$.

So, Unsat(ψ) and Valid(φ)

U-resolvent example

- Let *C* be a clause; if we negate all literals in *C*, we get C^-
- A unifier for a clause $C=\{l_1,\ldots,l_n\}$ is a unifier for $\{(l_1,l_2), (l_2, l_3), \ldots, (l_{n-1},l_n)\}$
- Let *C, D* be clauses (assume there are no common variables since we can rename vars). *K* is a U-resolvent of *C*, *D* iff there are non-empty $\underline{C'}\subseteq C$, $\underline{D'}\subseteq D$ s.t. σ is a unifier for $\underline{C'}\cup\underline{D'}^-$ and $K=(C\backslash\underline{C'} \cup D\backslash\underline{D'})\sigma$. Note $|\underline{C'}|$, $|\underline{D'}|$ can be >1
- Try this: $C = \{\neg S(c,x), \neg S(x,x)\},\ D = \{S(x,x), S(c,x)\}$

One possible U-resolution step

$$\begin{tikzcd} \{\underline{\neg S(c,x)}, \neg S(x,x)\} \arrow[dr, no head] & & \{\underline{S(c,y)}, S(y,y)\} \arrow[dl, no head, "\sigma = y \leftarrow x"] \\ & \{\neg S(x,x), S(x,x)\} & \end{tikzcd}$$

Tautology, so useless

U-resolvent example

- Let C be a clause; if we negate all literals in C, we get C^-
- A unifier for a clause $C=\{l_1,\dots,l_n\}$ is a unifier for $\{(l_1,l_2), (l_2, l_3), \dots, (l_{n-1},l_n)\}$
- Let C, D be clauses (assume there are no common variables since we can rename vars). K is a U-resolvent of C, D iff there are non-empty $\underline{C}'\subseteq C$, $\underline{D}'\subseteq D$ s.t. σ is a unifier for $\underline{C}'\cup\underline{D}'^-$ and $K=(C\setminus\underline{C}' \cup D\setminus\underline{D}')\sigma$. Note $|\underline{C}'|$, $|\underline{D}'|$ can be >1
- Try this: $C = \{\neg S(c,x), \neg S(x,x)\},\ D = \{S(x,x), S(c,x)\}$

$$\begin{tikzcd}
\{\underline{\neg S(c,x)}, \neg S(x,x)\} \arrow[dr, dash] & & \{\underline{S(c,y)}, S(y,y)\} \arrow[dl, dash, "\sigma = y \leftarrow x"] \\
& \{\neg S(x,x), S(x,x)\} &
\end{tikzcd}$$

$$\begin{tikzcd}
\{\neg S(c,x), \underline{\neg S(x,x)}\} \arrow[dr, dash] & & \{\underline{S(c,y)}, S(y,y)\} \arrow[dl, dash, "\sigma = x, y \leftarrow c"] \\
& \{\neg S(c,c), S(c,c)\} &
\end{tikzcd}$$

$$\begin{tikzcd}
\{\underline{\neg S(c,x)}, \neg S(x,x)\} \arrow[dr, dash] & & \{S(c,y), \underline{S(y,y)}\} \arrow[dl, dash, "\sigma = x, y \leftarrow c"] \\
& \{\neg S(c,c), S(c,c)\} &
\end{tikzcd}$$

$$\begin{tikzcd}
\{\neg S(c,x), \underline{\neg S(x,x)}\} \arrow[dr, dash] & & \{S(c,y), \underline{S(y,y)}\} \arrow[dl, dash, "\sigma = y \leftarrow x"] \\
& \{\neg S(c,x), S(c,x)\} &
\end{tikzcd}$$

All are tautologies
(useless)

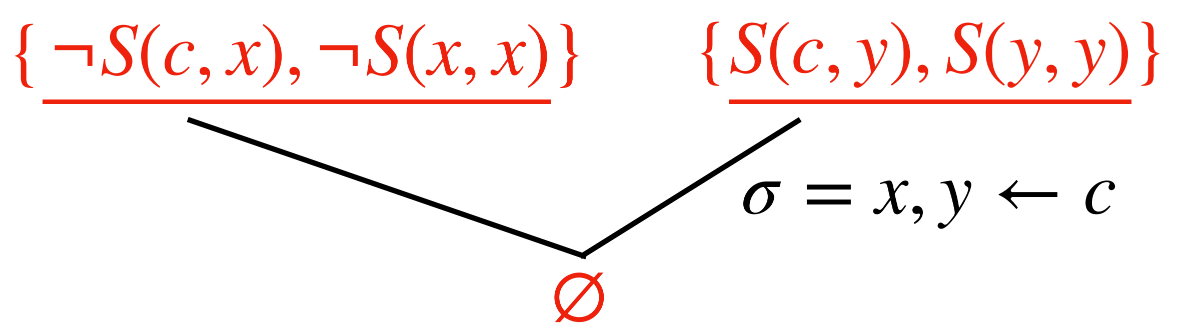
U-resolvent example

- Let *C* be a clause; if we negate all literals in *C*, we get C^-
- A unifier for a clause $C=\{l_1,\ldots,l_n\}$ is a unifier for $\{(l_1,l_2), (l_2, l_3), \ldots, (l_{n-1},l_n)\}$
- Let *C, D* be clauses (assume there are no common variables since we can rename vars). *K* is a U-resolvent of *C*, *D* iff there are non-empty $\underline{C}' \subseteq C$, $\underline{D}' \subseteq D$ s.t. σ is a unifier for $\underline{C}' \cup \underline{D}'^-$ and $K=(C \backslash \underline{C}' \cup D \backslash \underline{D}')\sigma$. Note $|\underline{C}'|$, $|\underline{D}'|$ can be >1
- Try this: $C = \{\neg S(c,x), \neg S(x,x)\},\ D = \{S(x,x), S(c,x)\}$

$$\begin{tikzcd}
{\{\underline{\neg S(c,x), \neg S(x,x)}\}} \arrow[dr, dash] & & {\{\underline{S(c,y)}, S(y,y)\}} \arrow[dl, dash, "\sigma = x, y \leftarrow c"] & & {\{\neg S(c,x), \underline{\neg S(x,x)}\}} \arrow[dr, dash] & & {\{\underline{S(c,y), S(y,y)}\}} \arrow[dl, dash, "\sigma = x, y \leftarrow c"] \\
& {\{S(c,c)\}} \arrow[drr, dash] & & & & {\{\neg S(c,c)\}} \arrow[dll, dash, "\sigma = \iota \text{ the identity substitution}"] & \\
& & & \varnothing & & &
\end{tikzcd}$$

U-resolvent example

- Let C be a clause; if we negate all literals in C, we get C^-
- A unifier for a clause $C=\{l_1,\ldots,l_n\}$ is a unifier for $\{(l_1,l_2), (l_2, l_3), \ldots, (l_{n-1},l_n)\}$
- Let C, D be clauses (assume there are no common variables since we can rename vars). K is a U-resolvent of C, D iff there are non-empty $\underline{C'}\subseteq C$, $\underline{D'}\subseteq D$ s.t. σ is a unifier for $\underline{C'}\cup\underline{D'}^-$ and $K=(C\backslash\underline{C'} \cup D\backslash\underline{D'})\sigma$. Note $|\underline{C'}|$, $|\underline{D'}|$ can be >1
- Try this: $C = \{\neg S(c,x), \neg S(x,x)\},\ D = \{S(x,x), S(c,x)\}$



$\{\underline{\neg S(c,x), \neg S(x,x)}\}$
$\{\underline{S(c,y), S(y,y)}\}$
$\sigma = x,y \leftarrow c$
$\varnothing$

- This is the Barber of Seville problem: Prove that there is no barber who shaves all those, and those only, who do not shave themselves.

$$\neg\langle \exists b\ \langle \forall x\ S(b,x) \equiv \neg S(x,x)\rangle\rangle$$

Schedule

- 11/8: FOL/SMT
- ~~11/11: Temporal Logic/ Safety & Liveness/ Buchi~~ (Veteran's Day)
- 11/15: Refinement
- 11/18: Paper Presentations
- 11/22: Paper Presentations
- 11/29: Term Rewriting
- 12/2: Projects, Exam 2 (Take home)
- 12/6: Projects

Proof Theory

- $\Phi \vdash \phi$ denotes that ϕ is provable from Φ
- Provability should be machine checkable
- It may seem hopeless to nail down what a proof is
  - don't mathematicians expand their proof methods?
- FOL has a fairly simply set of obvious rules
- There are many equivalent ways of defining proof

Sequent Calculus

- A sequent is a nonempty sequence of formulas
- Sequent rules:

$$\frac{\Gamma\ \neg\phi\ \ \psi \qquad \Gamma\ \neg\phi\ \ \neg\psi}{\Gamma\ \ \phi} \qquad \frac{}{\Gamma\ \ \phi}\ \text{if } \phi \text{ is a member of } \Gamma$$

- The left rule says if you have a proof of both $\neg\psi$ and ψ from $\Gamma \cup \{\neg\phi\}$, that constitutes a proof of ϕ from Γ
- If there is a derivation of the sequent $\Gamma\ \phi$, then we write $\vdash \Gamma\ \phi$ and say that $\Gamma\ \phi$ is *derivable*
- A formula ϕ is *formally provable* or *derivable* from a set Φ of formulas, written $\Phi \vdash \phi$, iff there are *finitely* many formulas $\phi_1, \ldots, \phi_n$ in Φ s.t. $\vdash \phi_1 \ldots \phi_n\ \phi$

Sequent Rules

Antecedent Rule (Ant)

$$\frac{\Gamma \quad \varphi}{\Gamma' \quad \varphi} \text{ if every member of } \Gamma \text{ is also a member of } \Gamma'.$$

A sequent Γ ϕ is *correct* if Γ ⊨ ϕ

A rule is *correct*: applied to correct sequents, it yields correct sequents

Notice that the sequent rules are correct

Assumption Rule (Assm)

$$\frac{}{\Gamma \quad \varphi} \text{ if } \varphi \text{ is a member of } \Gamma.$$

Proof by Cases Rule (PC)

$$\begin{array}{lll} \Gamma & \psi & \varphi \\ \Gamma & \neg\psi & \varphi \\ \hline \Gamma & & \varphi \end{array}$$

Contradiction Rule (Ctr)

$$\begin{array}{lll} \Gamma & \neg\varphi & \psi \\ \Gamma & \neg\varphi & \neg\psi \\ \hline \Gamma & & \varphi \end{array}$$

Sequent Rules for ∨

∨-Rule for the Antecedent (∨ A)

$$\frac{\begin{array}{lll}\Gamma & \varphi & \xi \\ \Gamma & \psi & \xi\end{array}}{\begin{array}{lll}\Gamma & (\varphi \vee \psi) & \xi\end{array}}$$

∨-Rule for the Succedent (∨ S)

$$\text{(a)}\frac{\Gamma \quad \varphi}{\Gamma \quad (\varphi \vee \psi)} \qquad \text{(b)}\frac{\Gamma \quad \varphi}{\Gamma \quad (\psi \vee \varphi)}$$

Derived Sequent Rules

Tertium non datur (Ctr)

$$\frac{}{(\varphi \vee \neg\varphi)}$$

Proof? We can prove it by assuming φ, getting $\varphi \vee \neg\varphi$ and similarly with $\neg\varphi$.

1.	φ	φ	(Ant)
2.	φ	$(\varphi \vee \neg\varphi)$	($\vee$ S)
3.	$\neg\varphi$	$\neg\varphi$	(Ant)
4.	$\neg\varphi$	$(\varphi \vee \neg\varphi)$	($\vee$ S)
5.		$(\varphi \vee \neg\varphi)$	(PC)

Sequent Rules

Reflexivity Rule for Equality ($\equiv$)

$$\frac{}{t \equiv t}$$

Substitution Rule for Equality (Sub)

$$\frac{\Gamma \qquad\qquad \varphi\frac{t}{x}}{\Gamma \quad t \equiv t' \quad \varphi\frac{t'}{x}}$$

Sequent Rules for ∃

∃-Introduction in the Succedent (∃ S)

$$\frac{\Gamma \quad \varphi\frac{t}{x}}{\Gamma \quad \exists x\varphi}$$

Proof Suppose $\Gamma \models \varphi\frac{t}{x}$. If $\mathcal{J} \models \Gamma$, we have $\mathcal{J} \models \varphi\frac{t}{x}$. By the substitution lemma, $\mathcal{J}\frac{\mathcal{J}.t}{x} \models \varphi$ and thus $\mathcal{J} \models \exists x\varphi$. □

∃-Introduction in the Antecedent (∃ A)

$$\frac{\Gamma \quad \varphi\frac{y}{x} \quad \psi}{\Gamma \quad \exists x\varphi \quad \psi} \text{ if } y \text{ is not free in } \Gamma\ \exists x\varphi\ \psi.$$

Proof So, $\Gamma\varphi\frac{y}{x} \models \psi$. Suppose $\mathcal{J} \models \Gamma$ and $\mathcal{J} \models \exists x\varphi$. Then there is an a such that $\mathcal{J}\frac{a}{x} \models \varphi$, but by the coincidence lemma, $(\mathcal{J}\frac{a}{y})\frac{a}{x} \models \varphi$. Since $\mathcal{J}\frac{a}{y}(y) = a$, we have $(\mathcal{J}\frac{a}{y})\frac{\mathcal{J}\frac{a}{y}(y)}{x} \models \varphi$ and by substitution lemma $\mathcal{J}\frac{a}{y} \models \varphi\frac{y}{x}$. Since $\mathcal{J} \models \Gamma$ and $y \notin \text{free}.\Gamma$, we get $\mathcal{J}\frac{a}{y} \models \Gamma$. Now, we get $\mathcal{J}\frac{a}{y} \models \psi$ and therefore $\mathcal{J} \models \psi$ because $y \notin \text{free}.\psi$. □

Gödel's Completeness Part 1

- For all Φ and ϕ, $\Phi \vdash \phi$ iff there is a finite $\Phi_0 \subseteq \Phi$ s.t. $\Phi_0 \vdash \phi$
  - Directly from definition of derivable
- Easy part of Gödel's completeness theorem
  - $\Phi \vdash \phi$ implies $\Phi \vDash \phi$
  - By induction on structure of derivations, using correctness of sequent rules
- Φ is *consistent*, written Con Φ, iff there is no formula ϕ such that $\Phi \vdash \phi$ and $\Phi \vdash \neg\phi$
- Φ is *inconsistent*, written Inc Φ, iff Φ is not consistent, i.e., there is a formula ϕ such that $\Phi \vdash \phi$ and $\Phi \vdash \neg\phi$
- Inc Φ iff for all ϕ, $\Phi \vdash \phi$
- Con Φ iff there is some ϕ s.t. not $\Phi \vdash \phi$
- For all Φ, Con Φ iff Con Φ_0 for all finite subsets Φ_0 of Φ

Consistency and SAT

- Sat Φ implies Con Φ
  - Inc $\Phi \Rightarrow \Phi \vdash \phi$ and $\Phi \vdash \neg\phi \Rightarrow \Phi \models \phi$ and $\Phi \models \neg\phi \Rightarrow$ not Sat Φ
- For all Φ and ϕ the following holds
  - **$\Phi \vdash \phi$ iff Inc $\Phi \cup \{\neg\phi\}$**
  - $\Phi \vdash \neg\phi$ iff Inc $\Phi \cup \{\phi\}$
  - If Con Φ, then Con $\Phi \cup \{\phi\}$ or Con $\Phi \cup \{\neg\phi\}$

Gödel's Completeness Theorem

- We have show the easy part of the completeness theorem
  - $\Phi \vdash \phi$ implies $\Phi \models \phi$
- What about the converse?
- Gödel's completeness theorem: $\Phi \models \phi$ implies $\Phi \vdash \phi$
- Lemma: Con Φ implies Sat Φ
- Φ is *consistent*, written Con Φ, iff there is no formula ϕ such that $\Phi \vdash \phi$ and $\Phi \vdash \neg\phi$
- Proof (of completeness): $\Phi \models \phi$

iff {previous lemma} not Sat $(\Phi \cup \{\neg\phi\})$

iff {above lemma, soundness} not Con $(\Phi \cup \{\neg\phi\})$

iff {previous slide} $\Phi \vdash \phi$

Gödel's Completeness Theorem

- $\Phi \vdash \phi$ iff $\Phi \vDash \phi$
- What does this mean for group theory?
- What about new proof techniques?
- Once we show the equivalence between $\vdash \phi$ and $\vDash$, we can transfer properties of one to the other
  - Compactness theorem:
    (a) $\Phi \vDash \phi$ iff there is a finite $\Phi_0 \subseteq \Phi$ such that $\Phi_0 \vDash \phi$
    (b) Sat Φ iff for all finite $\Phi_0 \subseteq \Phi$, Sat Φ_0
- From the proof, we get the Löwenheim-Skolem theorem: Every satisfiable and at most countable set of formulas is satisfiable over a domain which is at most countable

Gödel's 1st Incompleteness Theorem

- A set is *recursive* iff $\in$ can be decided by a Turing machine
- Assuming Con(ZF), the set $\{\phi : ZF \vdash \phi\}$ is not recursive
- More generally, for any consistent extension C of ZF:
  - $\{\phi : C \vdash \phi\}$ is not recursive
  - Intuitively clear: embed Turing machines in set theory
  - Encode halting problem! as a formula in set theory
- Theorem: If C is a recursive consistent extension of ZF, then it is incomplete, i.e., there is a formula ϕ such that $C \nvdash \phi$ and $C \nvdash \neg\phi$
- Proof Outline: If not, then for every ϕ, either $C \vdash \phi$ or $C \vdash \neg\phi$. We can now decide $C \vdash \phi$: enumerate all proofs of C. Stop when a proof for ϕ or $\neg\phi$ is found

FOL Observations

- In ZF, the axiom of choice is neither provable nor refutable
- In ZFC, the continuum hypothesis is neither provable nor refutable
- By Gödel’s first incompleteness theorem, no matter how we extend ZFC, there will always be sentences which are neither provable nor refutable
- There are non-standard models of $\mathbb{N}$, $\mathbb{R}$ (un/countable)
- Since any reasonable proof theory has to be decidable, and TMs can be formalized in FOL (set theory), any logic can be reduced to FOL
- Building reliable computing systems requires having programs that can reason about other programs and this means we have to really understand what a proof is so that we can program a computer to do it

Non-Standard Models

- Let $N_s = \langle \omega, s, 0 \rangle$, where s is the successor function. N_s satisfies:
  - (the successor of any number differs from that number) $\langle \forall x\ x \neq s(x) \rangle$
  - (s is injective) $\langle \forall x,y\ x \neq y \Rightarrow s(x) \neq s(y) \rangle$
  - (every non-0 number has a predecessor) $\langle \forall x\ x \neq 0 \Rightarrow \langle \exists y\ x = s(y) \rangle \rangle$
- Let $\Psi = \text{Th } N_s \cup \{x \neq 0, x \neq s(0), \ldots, x \neq s^n(0), \ldots\}$
- Every finite subset of Ψ has a model, so Ψ has a model (compactness)
- By Lowenheim-Skolem, let $\mathfrak{U}$ be a countable model of Ψ
  - $\mathfrak{U}$ includes $0, s(0), \ldots, s^n(0), \ldots$, and a, a non-standard number
  - a has a successor, predecessor, and they have successors, predecessors
  - so a is part of a $\mathbb{Z}$-chain
  - hence, there is a countable model, $\mathfrak{U}$, which is *not* isomorphic to N_s
- While there is a complete axiomatization for Th N_s, once the logic is powerful enough (add +, *, <), completeness goes out the window

$0, s(0), \ldots, s^n(0), \ldots, \qquad \ldots, p^n(a), \ldots, p(a), a, s(a), \ldots, s^n(a), \ldots$

$\mathbb{Z}$-chain

$p(a)$ is the predecessor of a

(isomophic to $\mathbb{Z}$)

First Order Theories

- *Signature* Σ: set of constant, function, predicate symbols
- Σ-*term*, Σ-*atom*, Σ-*literal*, Σ-*formula*, Σ-*sentence*
- Σ-*interpretation* assigns meaning to vars, Σ symbols, formulas
- Σ-*theory* is a set of Σ sentences
- For Σ-theory T, a *T-interpretation* satisfies all sentences in T
- *Validity problem* for T: is φT-valid (true in all T-interpretations)?
- *Satisfiability problem*: is φT-sat (true in some T-interpretation)?
- *Quantifier free* versions of decision problems
- Decision problem is *decidable* if there is a decision procedure

First Order Theories

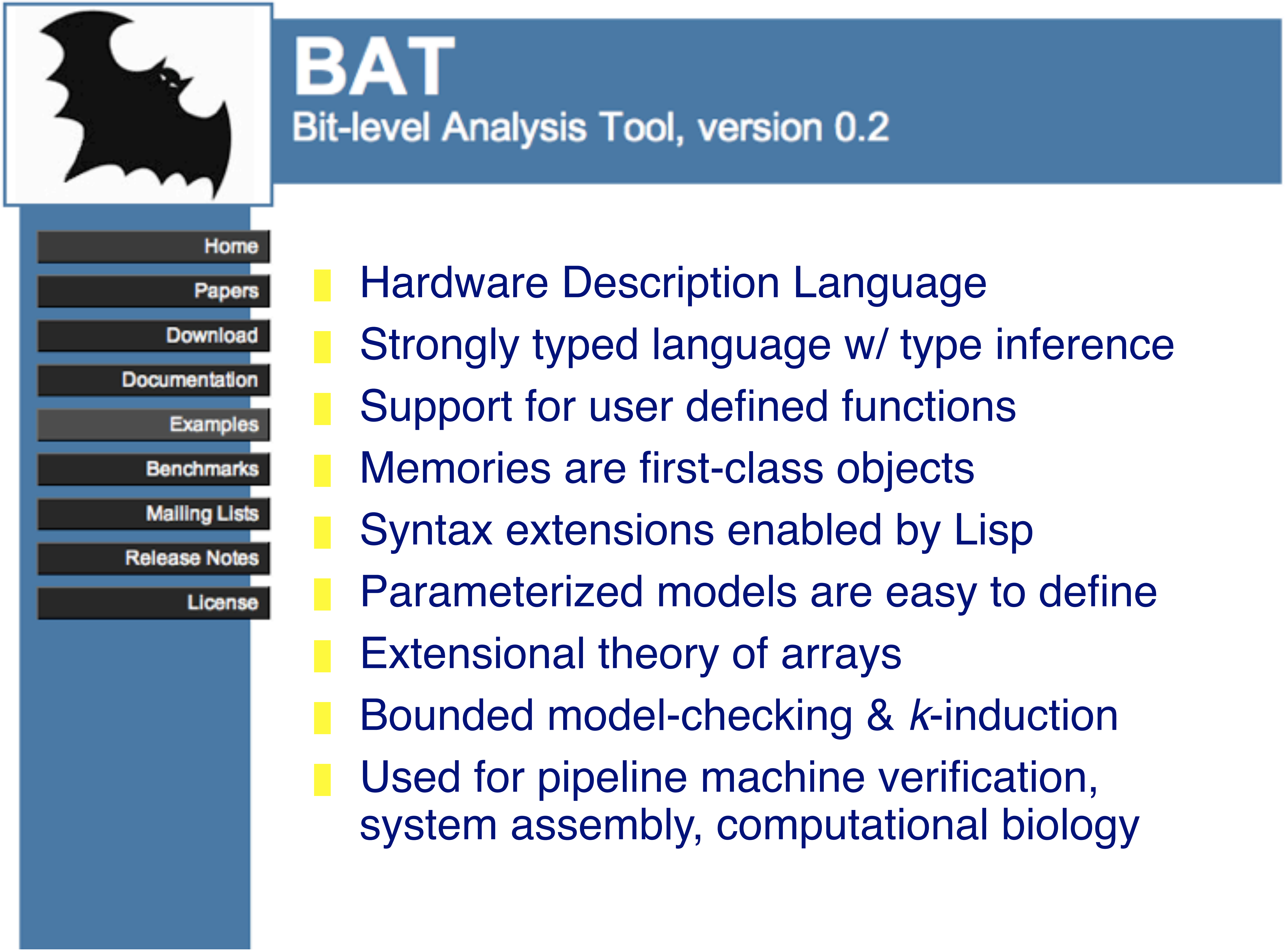
- Theory of equality: $\Sigma_=$ = FOL symbols, empty theory
  - Validity problem undecidable (FOL)
  - Quantifier-free validity problem decidable (congruence closure)
- Theory of arrays: Σ_A = {read, write}, array axioms
  - Validity problem undecidable
  - Quantifier-free validity problem decidable
- Theory of lists, Σ_L = (cons, car, cdr), list axioms
  - Validity problem decidable (Oppen) not elementary
  - Quantifier-free satisfiability solvable in linear time

First Order Theories

- Theory of integers, $\Sigma_{\mathbb{Z}}$ = (+, -, ≤, constants), all true sentences
  - Validity problem decidable (Presburger 1929) 3EXP (Cooper)
  - Quantifier-free satisfiability NP-complete (ILP) (Papadimitriou)
  - Adding × leads to undecidability even quantifier-free (Matiyasevich)
- Theory of reals, $\Sigma_{\mathbb{R}}$ = ($\Sigma_{\mathbb{Z}}$, rational constants), all true sentences
  - Validity problem decidable 2EXP (Ferrante and Rackoff)
  - Quantifier-free satisfiability problem in P (Khachiyan)
  - Adding × is still decidable (Tarski) 2EXP (Collins)

Satisfiability Modulo Theories

- Enabling technology: improved SAT solvers (CDCL)
- Eager methods: compile to SAT
  - Bryant et. al., Pnueli, Strichman, ...
  - Systems: UCLID [LS04], BAT [MVS07]
  - Sometimes this is the best option
- Lazy methods:
  - SAT solver is used to orchestrate theory cooperation
  - Barrett, Cimatti, Dill, deMoura, Ruess, Stump, ...
  - Systems: ICS[F..01], CVC [BDS02], MathSAT[A..02],...

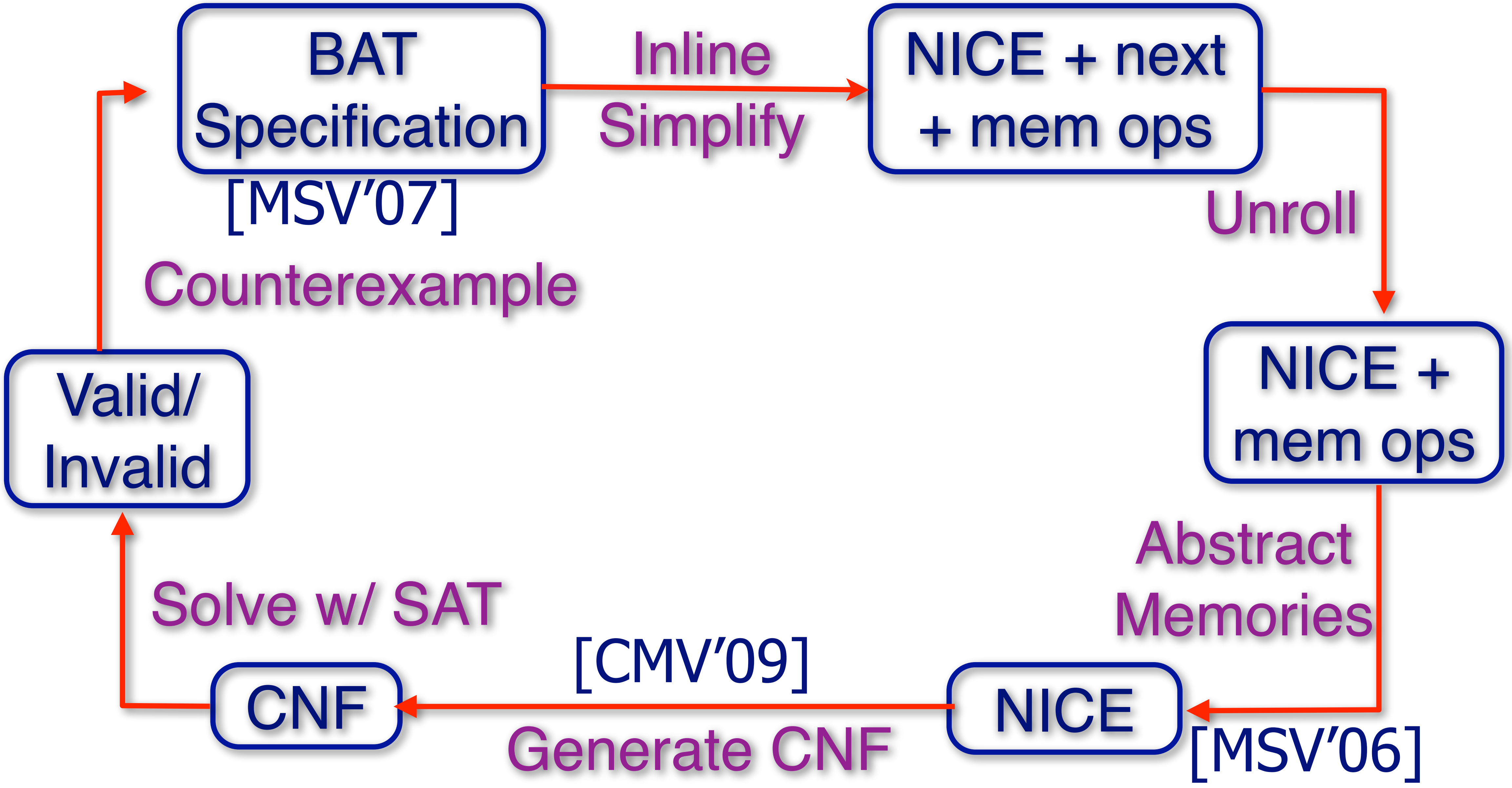


BAT

Bit-level Analysis Tool, version 0.2

- Hardware Description Language
- Strongly typed language w/ type inference
- Support for user defined functions
- Memories are first-class objects
- Syntax extensions enabled by Lisp
- Parameterized models are easy to define
- Extensional theory of arrays
- Bounded model-checking & *k*-induction
- Used for pipeline machine verification, system assembly, computational biology

BAT Decision Procedure


$$\begin{tikzcd}
\boxed{\text{BAT Specification}} \arrow[r, "\text{Inline}", "\text{Simplify}"'] & \boxed{\text{NICE + next + mem ops}} \arrow[d, "\text{Unroll}"] \\
\boxed{\text{Valid/ Invalid}} \arrow[u, "\text{Counterexample}"] & \boxed{\text{NICE + mem ops}} \arrow[d, "\text{Abstract Memories}"] \\
\boxed{\text{CNF}} \arrow[u, "\text{Solve w/ SAT}"] & \boxed{\text{NICE}} \arrow[l, "\text{Generate CNF}"]
\end{tikzcd}$$

BAT Memory Abstraction

Extensional theory of arrays:
Memories are treated as first class objects.

$$
\begin{array}{l}
\texttt{(= (set } m_1\ a_1\ v_1\texttt{)} \\
\quad\ \ \texttt{(set } m_2\ a_1\ v_2\texttt{))}
\end{array}
$$

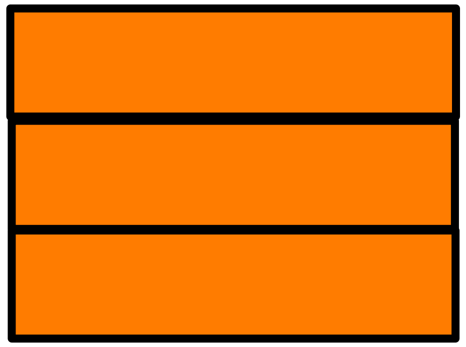
Memories can be directly compared in all contexts.

$$
\begin{array}{l}
\texttt{(not (= (set } m_1\ a_1\ v_1\texttt{)} \\
\qquad\qquad\ \ \texttt{(set } m_2\ a_1\ v_2\texttt{)))}
\end{array}
$$

BAT Memory Abstraction

`(get (set (set m a`$_1$` v`$_1$`) a`$_2$` v`$_2$`) a`$_3$`)`

Abstracted memory



- Determine number of unique gets and sets (*n*).
- Generate abstract memory consisting of *n* words.
- Apply abstraction to original addresses.
- Note: size of abstract addresses is lg(*n*).

Combining Decision Procedures

- Pioneers
  - Nelson-Oppen combination method [1979]
  - Nelson-Oppen congruence closure procedure [1980]
  - Shostak combination method [1984]
  - Integrating Decision Procedures into Theorem Provers [1988]
- Systems
  - Nqthm [BM 1997]
  - Simplify [DNS 2005]

Nelson-Oppen Method

- Decide satisfiability of quantifier-free φ over Σ_1 and Σ_2
- Convert into a conjunction of literals (DNF)
- Purify: convert into a conjunction $\Gamma_1 \cup \Gamma_2$ s.t.
  - each literal in Γ_i is a Σ_i literal
  - $\Gamma_1 \cup \Gamma_2$ is $\Sigma_1 \cup \Sigma_2$ SAT iff φ is
- Check: For each equivalence *E* over shared vars *V*
  - $\Gamma_i \cup \alpha(V,E)$ is T_i-SAT
  - $\alpha(V,E) = \{x=y : xEy\} \cup \{x \neq y : \text{not } xEy\}$ (arrangement)
- If there is such an equivalence, SAT, else UNSAT
- Can extend to many theories

Example

- $0 \leq x \land x \leq 1 \land f(x) \neq f(1) \land f(x) \neq f(0)$
- Purification?
  - $\Gamma_{\mathbb{Z}} = 0 \leq x \land x \leq 1 \land u=1 \land v=0$
  - $\Gamma_{=} = f(x) \neq f(u) \land f(x) \neq f(v)$
- Shared variables S = {x, u, v}, so 5 arrangements
- SAT?
- For all arrangements over S we have $T_{\mathbb{Z}}$ or $T_{=}$ unsat

Nelson-Oppen Method

- Disjoint signatures Σ_1, Σ_2
- T_1, T_2 decidable and *stably infinite*
  - For every T-satisfiable quantifier-free φ there exists a T-interpretation with an infinite domain satisfying φ
- $T_{\mathbb{R}}$, $T_{\mathbb{Z}}$, $T_{=}$, T_A, and T_L are all stably infinite.
- T= {(∀x : x=a ∨ x=b)} is not stably infinite.
- a=b ∧ f(c) ≠ f(d) is T-Unsat, yet NO method says Sat
- Complexity: How many equivalences? Bell number
  - If T_1, T_2 in NP, so is the combined decision procedure