Lecture 14

Pete Manolios
Northeastern

Discussion

- Exams returned
- Safety, liveness
- Refinement
- Hardware verification
- Refinement
- Temporal Logic
- Systems Verification Day
- Commercial Verification Tools
- Etc

Question 3

```
(A (A (R (A x y)) z) w)
= { R1 }
(A (A (A (R y) (R x)) z) w)
= { R2 }
(A (A (A (R x) (R y)) z) w)
= { R3 }
(A (A (R x) (A (R y) z)) w)
= { R2 }
(A (A (R x) (A z (R y))) w)
= { R3 }
(A (R x) (A (A z (R y)) w))
= { R3 }
(A (R x) (A z (A (R y) w)))
= { R2 }
(A (R x) (A z (A w (R y))))
```

```
R1. (R (A x y)) = (A (R y) (R x))
R2. (A y x) = (A x y)
R3. (A (A x y) z) = (A x (A y z))
```

Rewriting is the most important part of ACL2, so remember:

1. *Left to right*
2. *Inside-out*
3. *Reverse chronological*

Plus special handling of permutative rules, type reasoning, linear arithmetic, tau, conditional rewriting, forward chaining, … (most of which I didn't test)

FOL Checking

- 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. $g \subseteq G_n$. If $\exists n$ s.t. Unsat G_n, then Unsat ψ and Valid ϕ
- Question 1: SAT checking
  - Gilmore (1960): Maintain conjunction of instances so far in DNF, so SAT checking is easy, but there is a blowup due to DNF
  - Davis Putnam (1960): Convert ψ to CNF, so adding new instances does not lead to blowup
  - In general, any SAT solver can be used, eg, DPLL much better than DNF
- Question 2: How should we generate G_i?
  - Gilmore: Instances over terms with at most 0, 1, … , functions
  - Any such “naive” method leads to lots of useless work, eg, the book has code for minimizing instances and reductions can be drastic

Unification

- Better idea: intelligently instantiate formulas. Consider the clauses

  $\{P(x, f(y)) \vee Q(x, y), \neg P(g(u), v)\}$
- Instead of blindly instantiating, use $x=g(u)$, $v=f(y)$ so that we can resolve

  $\{P(g(u), f(y)) \vee Q(g(u), y), \neg P(g(u), f(y))\}$
- Now, resolution gives us

  $\{Q(g(u), y)\}$
- Much better than waiting for our enumeration to allow some resolutions
- Unification: Given a set of pairs of terms $S = \{(s_1,t_1), \ldots, (s_n,t_n)\}$ a *unifier* of S is a substitution σ such that $s_i|\sigma = t_i|\sigma$
- We want an algorithm that finds a *most general* unifier if it exists
  - σ is *more general* than τ, $\sigma \leq \tau$, iff $\tau = \delta \circ \sigma$ for some substitution δ
  - Notice that if σ is a unifier, so is $\tau \circ \sigma$
- Similar to solving a set of simultaneous equations, e.g., find unifiers for
  - $\{(P(f(w), f(y)), P(x, f(g(u))), (P(x,u), P(v,g(v))\}$ and $\{(x, f(y)), (y, g(x))\}$

Using Unification

- Assume we have a unification algorithm. How do we use it?
- Consider DP. When we instantiate a set of clauses, say
  $\{P(x, f(y)) \vee Q(x, y), \neg P(g(u), v)\}\sigma, \quad \sigma = \{x \leftarrow g(u), u \leftarrow f(y)\}$
- We obtain
  $\{P(g(u), f(y)) \vee Q(g(u), y), \neg P(g(u), f(y))\}$
- The original clauses state
  $\langle \forall x, y, u, v \ (P(x, f(y)) \vee Q(x, y)) \ \wedge \ \neg P(g(u), v)\rangle$
- The instantiated clauses are implied because they state
  $\langle \forall u, y \ (P(g(u), f(y)) \vee Q(g(u), y)) \ \wedge \ \neg P(g(u), f(y))\rangle$
- Notice that we are free to further instantiate the above instantiated clauses
- In contrast, if we use DPLL and case split, then we have to be careful, e.g., if we first assume $P(x,f(y))$ and then $Q(x,y)$, then in subsequent instantiations, x and y have to be instantiated the same way because
  $\langle \forall x, y \ P(x, f(y)) \vee Q(x, y)\rangle \not\Rightarrow \langle \forall x, y \ P(x, f(y))\rangle \vee \langle \forall x, y \ Q(x, y)\rangle$
- DP is *local* or *bottom-up*, whereas DPLL is *global* or *top-down*

Unification Basics

- Unification Problem: Given a set of pairs of terms $S = \{(s_1,t_1), \ldots, (s_n,t_n)\}$ a *unifier* of S is a substitution σ such that $s_i|\sigma = t_i|\sigma$ (we'll write $s_i\sigma = t_i\sigma$)
- $U(S)$ is the set of all unifiers of S; notice that if σ is a unifier, so is $\tau \circ \sigma$
- σ is *more general* than τ, $\sigma \leq \tau$, iff $\tau = \delta\sigma$ $(\delta \circ \sigma)$ for some substitution δ
- $\leq$ is a preorder; let δ be the identify for reflexivity
  - transitivity: if $\sigma \leq \tau$, $\tau \leq \theta$ then $\tau = \delta\sigma$, $\theta = \gamma\tau = \gamma(\delta\sigma) = (\gamma\delta)\sigma$
  - $\sigma \sim \tau$ iff $\sigma \leq \tau$, $\tau \leq \sigma$. Notice that if $\sigma = x \leftarrow y$, $\tau = y \leftarrow x$, then $\sigma \sim \tau$
  - $\sigma \sim \tau$ iff there is a *renaming* (bijection on Vars) θ s.t. $\sigma = \theta\tau$
- A *most general unifier* (mgu) is $\sigma \in U(S)$ s.t. for all $\tau \in U(S)$, $\sigma \leq \tau$
  - What is an mgu for $x=y$? $x \leftarrow y$? $y \leftarrow x$? $x \leftarrow z, y \leftarrow z$? $x \leftarrow y, z \leftarrow w, w \leftarrow z$?
- A substitution is *idempotent* if $\sigma\sigma = \sigma$ (rules out last case above)
  - σ is idempotent iff Domain(σ) is disjoint from Vars(Range(σ))
- If a unification problem has a solution, then it has an idempotent mgu
- We want an algorithm that finds an mgu, if a unifier exists