Randomized Model Finder

Everything that could lead us to solve the paradox...

Model Finding Basics

- First Order Logic Formula
  - Predicate
  - Functions
- Interpretation
  - (Finite) Domain
  - Interpretation of predicates and functions
- Model: Interpretation that satisfies some FOL formulas

Finding Models...

- Exhaustive search
- SEM: Search using constraint propagation method
- MACE: Translating « instanciated » FOL formulas into propositional clauses, solved by a SAT-Solver
- KODKOD: Takes into account partial instance

MACE

- Reduction FOL => Propositional Logic
  1. Propositional Encoding
  2. Flattening
  3. Instanciating
- Solve the SAT problem

Flattening

- Translate all FOL clauses into clauses containing only shallow literal
  - $P(x,\dots,y)$ or $\neg P(x,\dots,y)$
  - $f(x,\dots,y) = z$ or $f(x,\dots,y) \neq z$
  - $x = y$
- Example:

  $P(a,f(x))$ leads to

  $a \neq y \mid f(x) \neq z \mid P(y,z)$

Instanciation

- Instances
  - Instanciate every free variable with each domain element
- Functional Definitions
  - Express the requirement that a function has to give back the same value for the same arguments.
  - $(f(d) \neq x \mid f(d) \neq y)$ & ...
- Totality Definitions
  - $f(d) = 1 \mid \ldots \mid f(d) = s$

Paradox

- The number of clauses is growing exponentially with the number of variables: $|\text{domain}|^{\#\text{variables}}$
- Even worse: Flattening introduces a lot of auxiliary variables...
- Paradox is all about techniques for making the life of SAT-Solvers easier...

The need for speed

- Overview of optimizations
  - Reducing #Variables in Clauses (Splitting)
  - Incremental Search
  - Static Symmetry Reduction
  - Sort Inference

Splitting

- # instances needed for a clause is exponential to # variables in the clause
- More clauses with fewer variables is thus better
- { P(x,y) | Q(x,z) } can be split to
  { P(x,y) | S(x) } & { !S(x) | Q(x,z) }

Splitting

Let a clause $C[\alpha] \cup D[\beta]$
C and D are a proper binary split $\iff \exists x.(x \in \alpha \wedge x \notin \beta) \wedge \exists y.(y \in \beta \wedge y \notin \alpha)$

$$\{S(\alpha \cap \beta)\} \cup C[\alpha]$$

$$\{\neg S(\alpha \cap \beta)\} \cup D[\beta]$$

Splitting

- Repeating binary splits are possible, but greedy choices might destroy better later ones
- Paradox uses a simple heuristic
  - Least connected variable is split
  - Finds all possible splits, but does not necessarily lead to optimal split

Incremental Search

- Paradox uses several iterations with increasing domain size
- *Conflict Learning:* contradictions are converted into *learning clauses* and forwarded to the next iteration

Incremental Satisfiability

- Given the SAT instance for domain size s, for domain size s+1:
  - For *Instances* and *Function Definitions*, we can keep the previous clauses and add new ones
  - For *Totality Definitions*, clauses have to be replaced

Incremental Satisfiability

- Add a propositional variable d_s for each domain size s
- Adding $\neg d_s$ as a literal to each totality clause

Static Symmetry Reduction

- Due to the encoding in SAT, for each model all isomorphic variations are also models
- This is a problem for the SAT solver since SEM-style methods use Symmetry Reduction Techniques to reduce the search space
- Paradox thus adds constraints to remove symmetries statically

Sort Inference

- Think of 'sorts' as types
- 'sorted' models are easier to find
- Paradox tries to infer 'sorts' on the initially unsorted problems

And ?

- Within 2 min, Paradox is able to solve 90% of TPTP satisfiable problems. (Better than the previous CASC winner with a limit of 5 min)
- Within 10 min, Paradox solved for the first time 28 TPTP problems (including 15 open / unknown problems)

Our project...

- Goal: finding models in a randomized fashion
- Parse formulas in TPTP format
- Evaluate an interpretation against formulas
- So far, interpretations are generated using exhaustive search...
- Implemented in Scala: Stack Overflow problems