Automatic Synthesis of Decision Procedures: a Case Study of Ground and Linear Arithmetic

Predrag Janičić
Faculty of Mathematics, University of Belgrade
and
Alan Bundy
School of Informatics, University of Edinburgh

Calculemus 2007, RISC, Hagenberg, Austria, June 27–29, 2007.

Roadmap

- Decision Procedures and Bundy's Programme
- Method Generators
- Case Study: Ground Arithmetic
- Case Study: Linear Arithmetic
- Further Work and Conclusions

Roadmap

- Decision Procedures and Bundy's Programme
- Method Generators
- Case Study: Ground Arithmetic
- Case Study: Linear Arithmetic
- Further Work and Conclusions

Decision Procedures

- f is a decision procedure for a theory $\mathcal{T}$ if for any formula F it can tell whether or not $\mathcal{T} \vdash F$

- Many decision procedures available for many theories, also many combination schemes; often vital in theorem proving, explored in the context of SMT

- Difficult to develop and prone to implementation flaws, so automatic synthesis would be welcome

- Automatic synthesis would be important also for newly defined theories

Bundy's programme (1991) — Basic ideas

- Many steps in decision procedures and normalisation procedures are routine, often based on rewriting
- There are some families/kinds of such steps (e.g., remove, stratify, etc.)
- Many decision procedures are based on quantifier elimination
- The routine tasks in building decision procedures can be automated

Bundy's programme (1991) — Example

A stratify method can, by using the rules:

$$
\begin{array}{llll}
\texttt{st\_conj\_disj1}: & f_1 \wedge (f_2 \vee f_3) & \longrightarrow & (f_1 \wedge f_2) \vee (f_1 \wedge f_3) \\
\texttt{st\_conj\_disj2}: & (f_2 \vee f_3) \wedge f_1 & \longrightarrow & (f_2 \wedge f_1) \vee (f_3 \wedge f_1)
\end{array}
$$

transform a formula of the class

$$f \quad := \quad af|f \vee f|f \wedge f$$

into a formula of the (new) class f:

$$
\begin{array}{lll}
f & := & f'|f \vee f \\
f' & := & af|f' \wedge f'
\end{array}
$$

Bundy's programme (1991) — Further Steps

- Given several generated methods, it should be possible to combine these methods (automatically) into a compound method or, sometimes, into a DP for some theory

- For some normalisations and DPs successive rewritings are required; one is not enough (e.g., CNF)

- Methods (and compound methods) will be designed in such a way that their properties can be easily proved

- Building methods may require human assistance

Roadmap

- Decision Procedures and Bundy's Programme
- Method Generators
- Case Study: Ground Arithmetic
- Case Study: Linear Arithmetic
- Further Work and Conclusions

Method Generators

- Given an input BNF, a method kind, and rewrite rules, a method generator generates the output BNF and the corresponding method (in the spirit of proof planning)

- Normalisation methods are based on exhaustive application of rewrite rules. They transform formulae from one set to another set (e.g., into prenex normal form, DNF)

- Special-purpose method generators generate theory specific methods

Method Generators — Basic Normalisation Methods

- We implemented generators for several kinds of methods:
  - *Remove* for eliminating a certain symbol
  - *Stratify* for stratifying one syntactical class into two layers containing just *some* specific symbols
  - *Thin* for eliminating multiple occurrences of a unary symbol (e.g., elimination of multiple negations, by $\neg\neg x \longrightarrow x$)
  - *Absorb* for eliminating some recursion rules (e.g., $t ::= t \cdot real\_num | real\_num$ transforms to $t ::= real\_num$)
  - *Left-assoc* for reorganising within a class (e.g., for $\wedge$)

Method Generators — Special Purpose Generators

- Not of syntactical nature, theory specific (e.g., for linear arithmetic, generating a method for "cross-multiply-and-add")

- We implemented the following special-purpose generators:
  - for adjusting the innermost quantifier;
  - for generating *one-side methods*;
  - for isolating a variable;
  - for removing a variable.

Method Generators — Properties of Generated Methods

- Termination, soundness, completeness, are easily proved (from construction of the methods; of course, rewrite rules should be “sensible” w.r.t. the background theory (i.e., sound and complete))

- Slightly more difficult is some of the rewrite rules are conditional (some of the required statements can be proved by the generated procedures themselves)

Method Generators — Compound Method Generator

- Given method generators, an initial BNF and a set of rewrite rules, the initial BNF can be transformed step by step, yielding a sequence of methods (and BNFs), and reaching some goal BNF (e.g., a trivial one — consisting of $\perp$ and $\top$)

- The automated search engine
  - starts with the full BNF for a given theory
  - searches over all method generators and with all possible instantiations (arguments)
  - searches for a goal BNF

Method Generators — Compound Method Generator (2)

- Properties of this search engine:
  - search space is much smaller than if we searched over rewrite rules
  - the search is directed (and termination ensured) by a specific decreasing measure on the sequence of BNFs
  - the completeness can be ensured by iterative deepening
  - everything implemented in PROLOG in a system called ADEPTUS

Roadmap

- Decision Procedures and Bundy's Programme
- Method Generators
- Case Study: Ground Arithmetic
- Case Study: Linear Arithmetic
- Further Work and Conclusions

Ground arithmetic

- Ground arithmetic — no variables:

$$\begin{array}{rcl} f & := & af|\neg f|f \vee f|f \wedge f|f \Rightarrow f|f \Leftrightarrow f \\ af & := & \top|\bot|t = t|t < t|t > t|t \leq t|t \geq t|t \neq t \\ t & := & rc| - t|t \cdot t|t + t \end{array}$$

- We searched (over 59 necessary rewrite rules) for a compound method that can transform any formula into a formula described by: $f := \top|\bot$

- The search algorithm took $3s$; during that 48 methods were successfully generated, 22 of them in the final sequence

Ground arithmetic — Generated Decision Procedure

1. remove $\Leftrightarrow$
2. remove $\Rightarrow$
3. remove $\leq$
4. remove $\geq$
5. remove $\neq$
6. remove $>$
7. remove $-$
8. stratify $[\wedge, \vee]$
9. thin $\neg$
10. remove $\neg$
11. stratify $[\vee]$
12. stratify $[+]$
13. left_assoc $\vee$
14. left_assoc $+$
15. left_assoc $*$
16. absorb $*$
17. absorb $+$
18. remove $<$
19. remove $=$
20. left_assoc $\wedge$
21. remove $\wedge$
22. remove $\vee$

Roadmap

- Decision Procedures and Bundy's Programme
- Method Generators
- Case Study: Ground Arithmetic
- Case Study: Linear Arithmetic
- Further Work and Conclusions

Linear arithmetic

- Only addition (no multiplication, except multiplication by constants)

- Implementation prone to human flaws

- We searched (over 71 necessary rewrite rules) for a compound method that can transform any formula into a formula described by: $f := \top | \bot$

- The search algorithm took 5s; only 89 methods successfully generated, 51 of them in the final sequence — a Fourier-Motzkin-style procedure.

Roadmap

- Decision Procedures and Bundy's Programme
- Method Generators
- Case Study: Ground Arithmetic
- Case Study: Linear Arithmetic
- Further Work and Conclusions

Further Work

- Given an input BNF and a set of rewrite rules compute the output BNF (without having method generators)
- The output is not always definable by BNF (i.e., by a context-free grammar)
- This is subject of our current research

Conclusions

- Automatic/semi-automatic synthesis of DPs is possible
- While most of the approach is automated, some human assistance is required (for special-purpose method generators)
- All necessary rewrite rules have to be provided (the system complains if there are missing rules)

Conclusions (2)

- Formal properties of generated DPs are easily proved

- Reduced risk of human implementation flaws

- Synthesised procedures are structured and understandable to humans; rewrite rules are applied in stages

- The full system is implemented (ADEPTUS) and tested

Linear arithmetic (over reals) and Fourier/Motzkin's procedures

- uses series of transformations:
  - put the given formula into prenex normal form
  - eliminate $\Rightarrow$
  - put into disjunctive normal form, etc.
- "cross multiply and add step" (simplified):

$$(\exists x)(a < bx \ \wedge \ cx < d) \ \ (b, c > 0)$$

$$ac < bd$$

Proof planning and methods

- higher level reasoning
- methods are specification of tactics
- tactics give object level proofs
- a method has several slots: a name, input, preconditions, effect, output, postconditions and the name of the attached tactic.
- give structured proof-plans, understandable to humans

Proof planning and methods

- rewrite rules should be "sensible" w.r.t. the background theory (i.e., sound and complete); can be derived directly from axioms or from higher level statements

- example (for linear arithmetic):

$$(t_1 \geq t_2) \longrightarrow (t_2 < t_1 \ \vee \ t_1 = t_2)$$

Other Examples

- For fragments of the *Area method* for geometry

- For producing object level proofs for the SAT problem