Congruence Closure with Integer Offsets

Robert Nieuwenhuis and Albert Oliveras

UPC, Barcelona

LPAR, September 2003

Overview of this talk

1. Aim: solve SAT for the logic EUF (3 slides)
   - examples, complexity
   - applications, existing methods
2. Our approach: $DPLL(X)$ (1)
   - Prop. SAT methods: DP, DLL, DPLL (7 quick ones)
   - Chaff (1)
3. Congruence closure (CC) (11)
   - The problem. Applications.
   - Downey,Sethi,Tarjan 1980 JACM
   - Our approach for EUF: $DPLL(=)$
   - Initial Transformations
   - The algorithm for CC
   - CC with integer offsets
4. Final remarks

The logic EUF

Equality with Uninterpreted Functions: (Burch and Dill, 1994)
Ground first-order formulae with equality

Example 1: $a \neq c \;\vee\; b \neq d \;\;\vee\; f(a,b) = f(c,d)$
is valid (i.e. tautology)

Example 2: $f(f(f(a))) \neq b \wedge f(a) = a \wedge a = b$
is unsatisfiable

Example 3: $(\; P(a) \wedge \neg P(b) \;) \vee \; a \neq b$
is satisfiable, but $a = b$ falsifies it

Deciding satisfiability NP-complete.

The logic EUF (contd.)

Applications:

– Processor verification (Dill, Bryant et al.)

– (Finite) model finding in FOL

for consistency proofs, inductive theorem proving, CSP's ...

Example: there exist groups of card. 4 iff S is satisfiable:

Group axioms:

$$
\begin{aligned}
f(e,x) &= x \\
f(i(x),x) &= e \\
f(f(x,y),z) &= f(x,f(y,z))
\end{aligned}
$$

S has 4 new cts. a, b, c, d:

$a \neq b \wedge \ldots \wedge c \neq d$

$f(e,a) = a \wedge \ldots \wedge f(e,d) = d$

$f(i(a),a) = e \wedge \ldots$

$\ldots$

$e = a \vee e = b \vee e = c \vee e = d$

$f(a,a) = a \vee f(a,a) = b \ldots$

$\ldots$

EUF: current methods

Translate to propositional SAT and use DPLL:

–Bryant,German,Velev [ACM TOCL'01]

–MACE2 (McCune 1995)

–DDPP (Stickel 1994)

Specific techniques for finding FO models:

–Finder,SEM (Zhang,Zhang, 1995), Falcon (Zhang 96)

–MGTP (Hasegawa et al, 1992)

–MACE4 (McCune 2002)

Specific techniques for more general logics:

–Lemmas on Demand (de Moura, Ruess 2002)

Our approach: $DPLL(X)$

- No translation into propositional SAT
- Framework like $CLP(X)$ for SAT modulo theories
  (cf. related independent work by Cesare Tinelli [JELIA'02])
- **Use Davis-Putnam-Logemann-Loveland (DPLL) techniques à la Chaff (adapting some implementations we have)**
- Replace unit propagation by specialized incremental solvers.
  Example: EUF: congruence closure module in $DPLL(=)$.

Naive and less naive techniques for SAT

Notation: $\overline{1}4\overline{8}9$ denotes clause $\neg x_1 \vee x_4 \vee \neg x_8 \vee x_9$

Example:

$\overline{1}23$, 421, $7\overline{6}1$, $\overline{2}3\overline{1}$, $8\overline{3}\overline{1}$, $\overline{4}26$, $\overline{6}21$, $\overline{8}\overline{3}\overline{1}$, $6\overline{2}1$, 546, $\overline{7}\overline{6}1$

Truth table: 256 cases to be considered

Resolution:

$$\frac{x \vee C \qquad\qquad \neg x \vee D}{C \vee D}$$

Many (and big) clauses generated!

Ordered resolution: (e.g., $1 > 2 > \ldots > 8$)

Still too many clauses generated:

from $\overline{1}23 + 421$: 234
from $\overline{1}23 + 7\overline{6}1$: $237\overline{6}$
from $\overline{1}23 + \overline{6}21$: $23\overline{6}$

...

Methods for SAT (contd.)

Davis-Putnam 1960:

- Three rules used:
  1. Unit clause (one-literal clauses)
  2. Pure literal (only occurs with one sign)
  3. Resolution (after resolution between i and $\bar{i}$, eliminate clauses with occurrences of i or $\bar{i}$)
- Resolution produces quadratic growth of the input formula at each step

Methods for SAT (contd.)

Davis-Logemann-Loveland 1962:

- Rule 3 becomes Splitting rule: problem P produces two smaller problems: $P[x = 0]$ and $P[x = 1]$
- Method has the following features:
  1. Depth-first search with backtracking
  2. Low memory consumption
  3. Can decide splitting variable x on the fly, using heuristics with freedom for using different criteria on each branch!

Today this is usually called **DPLL**
(after Davis-Putnam-Logemann-Loveland).

Methods for SAT (contd.)

Example of DPLL:

$\overline{1}23$, 421, $7\overline{6}1$, $\overline{2}3\overline{1}$, $8\overline{31}$, $\overline{4}26$, $\overline{6}21$, $\overline{831}$, $6\overline{2}1$, 546, $\overline{76}1$

decision: $\overline{2}$

$\overline{1}23$, 421, $7\overline{6}1$, $8\overline{31}$, $\overline{4}26$, $\overline{6}21$, $\overline{831}$, 546, $\overline{76}1$

decision: $\overline{1}$

421, $7\overline{6}1$, $\overline{4}26$, $\overline{6}21$, 546, $\overline{76}1$

Propagation: 4, $\overline{6}$ Propagation: 6 Conflict!

Backtracking: we reverse decision $\overline{1}$

Methods for SAT (contd.)

Example of DPLL (contd.)

$\overline{1}23$, 421, $7\overline{6}1$, $\overline{2}3\overline{1}$, $8\overline{31}$, $\overline{4}26$, $\overline{6}21$, $\overline{831}$, $6\overline{2}1$, 546, $\overline{76}1$

decision: $\overline{2}$

$\overline{1}23$, 421, $7\overline{6}1$, $8\overline{31}$, $\overline{4}26$, $\overline{6}21$, $\overline{831}$, 546, $\overline{76}1$

decision: 1 (already flipped)

$\overline{1}23$, $8\overline{31}$, $\overline{4}26$, $\overline{831}$, 546,

Propagation: 3 Propagation: 8, $\overline{8}$ Conflict!

Backtracking: reverse decision $\overline{2}$

Methods for SAT (contd.)

Example of DPLL (contd.):

$\overline{1}23$, 421, $7\overline{6}1$, $\overline{2}3\overline{1}$, $8\overline{31}$, $\overline{4}26$, $\overline{6}21$, $\overline{831}$, $6\overline{2}1$, 546, $\overline{76}1$,

decision: 2 (already flipped)

$7\overline{6}1$, $\overline{2}3\overline{1}$, $8\overline{31}$, $\overline{831}$, $6\overline{2}1$, 546, $\overline{76}1$

decision: $\overline{1}$

$7\overline{6}1$, $6\overline{2}1$, 546, $\overline{76}1$

Propagation: 6 Propagation: 7, $\overline{7}$ Conflict!

Backtracking: reverse decision $\overline{1}$

Methods for SAT (contd.)

Example of DPLL (contd.):

$\overline{1}23$, 421, $7\overline{6}1$, $\overline{2}3\overline{1}$, $8\overline{31}$, $\overline{4}26$, $\overline{6}21$, $\overline{831}$, $6\overline{2}1$, 546, $\overline{76}1$

decision: 2 (already flipped)

$7\overline{6}1$, $\overline{2}3\overline{1}$, $8\overline{31}$, $\overline{831}$, $6\overline{2}1$, 546, $\overline{76}1$

decision: 1 (already flipped)

$\overline{2}3\overline{1}$, $8\overline{31}$, $\overline{831}$, 546,

Propagation: 3 Propagation: 8, $\overline{8}$ Conflict!

No backtracking pending: Unsatisfiable

Methods for SAT: Chaff

Malik et al (Princeton), 2001

- Excelent implementation of DPLL: 1-2 orders magn. faster
- Can handle many more problems from practice
- Combines ideas from previous systems
- Very efficient propagation mechanism:
  - 2 watched literals (non-false ones) in each clause
  - For each i, two linked lists: all watched lits. i, and all $\bar{i}$
  - When i becomes true, follow $\bar{i}$-list, searching in each clause another lit. to be watched.
    Propagate the other watched lit if there is none.
- Learning new clauses: exploits symmetry in real-world pbs.
- New heuristic for selecting next decision
- Restarts
- Other advanced systems, e.g., Forklift, Satzoo (SAT 2003 competition winners).

REMEMBER: Our approach: $DPLL(X)$

- No translation into propositional SAT
- Framework like $CLP(X)$ for SAT modulo theories
  (cf. related independent work by Cesare Tinelli [JELIA'02])
- Use DPLL techniques à la Chaff (adapting some implementations we have)
- **Replace unit propagation by specialized incremental solvers.**
  Example: **EUF: congruence closure module in** $DPLL(=)$.

Congruence closure

The problem: deduction in ground equational theories

Example:

$$\left.\begin{array}{rcl} f(a,g(a)) & = & g(b) \\ g(a) & = & h(a) \\ a & = & f(c,h(c)) \\ h(a) & = & a \\ c & = & h(h(h(a))) \end{array}\right\} \quad \models \quad a = g(b) \text{ ?}$$

- Decidable, Ackerman 1954
- $O(n\ log\ n)$ Downey,Sethi,Tarjan 1980 JACM
- See also: Kozen STOC'77, Nelson,Oppen JACM'80, Shostak JACM'84
- Many applications:
  - –compilers (common subexpresions),
  - –verification, deduction (combination of theories, ...)

Our approach for EUF: $DPLL(=)$

- Unit propagation: many calls to congruence closure (CC)
- $O(n\ log\ n)$ algorithm of Downey,Sethi,Tarjan:
  - requires initial transformations to graph of outdegree 2
  - heavily relies on pointers and sharing
  - not as clean as later abstract versions of CC: [Kapur97, BachmairTiwariVigneron00] (generally $O(n^2)$).
- Ground completion algorithms are $O(n^2)$ [PS96] or rely on classical $O(n\ log\ n)$ CC-algorithms [Snyder89]
- Our approach is $O(n\ log\ n)$ but clean and simple.
- Idea: two initial transformations at the formula level done in the $DPLL(=)$ framework once and for all on the initial EUF problem (not at each call to CC).

The two initial transformations:

1. Curryfy (like in the implementation of FP):
   - After Curryfying: only one binary symbol "$\cdot$" and constants.
   - Example: Curryfying $f(a, g(b), c)$ gives $\cdot(\cdot(\cdot(f, a), \cdot(g, b)), c)$

2. Flatten:
   - Allows one to assume: terms of depth ≤ 1
   - Introduces a linear number of new constants
   - Example: Flattening $\{ \cdot(\cdot(\cdot(f, a), \cdot(g, b)), c) = i\}$ gives

$$\{ \ \cdot(f, a) = d, \ \ \cdot(g, b) = e, \ \ \cdot(d, e) = h, \ \ \cdot(h, c) = i \ \}$$

After this:

- Literals in EUF formula between cts. only: $a = b$ or $a \neq b$
- Hidden inside the CC module there is a fixed set of equations E of the form $\cdot(a, b) = c$

Congruence closure: our view

Now the CC problem is: $E \models a = b?$ $(a, b, c, d, e$ cts.)
where in E there are only equations of the form $\cdot(c, d) = e$

Our data structures: (no union-find!)

1. Pending unions: a list of pairs of cts yet to be merged.
2. Representative table: array indexed by constants, with for each constant c its current representative $rep(c)$.
3. Class lists: for each repres., the list of all cts in its class.
4. Lookup table: for each input term $\cdot(a, b)$, $Lookup(rep(a), rep(b))$ returns in constant time a constant c such that $\cdot(a, b) = c$ ($\perp$ if there is none).
5. Use lists: for each representative a, the list of input equations $\cdot(b, c) = d$ such that a is $rep(b)$ or $rep(c)$ or both.

Congruence closure: our algorithm

Notation: c' means $rep(c)$

```
While Pending ≠ ∅ Do
  remove a = b from Pending
  If a' ≠ b' and, wlog., |ClassList(a')| ≤ |ClassList(b')| Then
    For each c in ClassList(a') Do
      set rep(c) to b' and add c to ClassList(b')
    EndFor
    For each ·(c, d) = e in UseList(a') Do
      If Lookup(c', d') is some f and f' ≠ e' Then
        add e' = f' to Pending
      EndIf
      set Lookup(c', d') to e'
      add ·(c, d) = e to UseList(b')
    EndFor
  EndIf
EndWhile
```

Congruence closure: our algorithm

Notation: c' means $rep(c)$

```
While Pending ≠ ∅ Do
  remove a = b from Pending
  If a' ≠ b' and, wlog., |ClassList(a')| ≤ |ClassList(b')| Then
    For each c in ClassList(a') Do
      set rep(c) to b' and add c to ClassList(b')
    EndFor
    For each ·(c, d) = e in UseList(a') Do
      If Lookup(c', d') is some f and f' ≠ e' Then
        add e' = f' to Pending
      EndIf
      set Lookup(c', d') to e'
      add ·(c, d) = e to UseList(b')
    EndFor
  EndIf
EndWhile
```

$a' = b'$

$\cdot(a', d') = e \qquad \cdot(b', d') = f$

Analysis of the algorithm

$O(n\ log\ n)$ time and linear space:

- assume k different constants (usually, $k \ll n$)
- each ct changes representative at most $log\ k$ times
- maintenance rep and $ClassList$: $k\ log\ k$
- maintentance $Lookup$ and $UseList$: $2n\ log\ k$

Correctness:

- Let $RepresentativeE$ be the non-trivial eqs $a = a'$ and $\cdot(a', b') = c'$ where a, b and c cts in E_0 and c is $Lookup(a', b')$.
- Note: final $RepresentativeE$ is the resulting closure (a convergent TRS)
- Key invariant: $(RepresentativeE \cup Pending)^* = E_0^*$

Experimental results

		Our algorithm	
File	**Abstract CC**	**No preprocess**	**Preprocess**
ex4211.5000	0.212547	0.039415	1.218427
ex4301.5000	3.720394	0.077519	1.409287
ex4301.6000	4.282850	0.092307	1.738292
ex4211.7000	0.270680	0.044061	1.641836
ex4301.7000	2.293164	0.107017	2.003430
ex4211.10000	0.357135	0.040921	2.365986
...			
TOTAL TIME	39.452168	0.738211	23.636892

CONCLUSION:Better performance, specially once preprocessed

Integer Offsets

- Bryant et al. add interpreted succ and pred symbols
- write (sub)terms $\underbrace{succ(\ldots succ}_{k\ times}(t)\ldots)$ as $t+k$

  same with negative k for $\underbrace{pred(\ldots pred}_{k\ times}(t)\ldots)$
- Example: $f(a)=c \wedge f(b+1)=c+1 \wedge a-1=b$

  Note that now E_0 can be unsatisfiable.
- $$\begin{array}{rcl} a+2 & = & b-3 \\ b-5 & = & c+7 \\ c & = & d-4 \end{array} \quad \text{is} \quad \begin{array}{rcl} a & = & b-5 \\ b & = & c+12 \\ c & = & d-4 \end{array}$$

  An infinite number of classes, the ones of $\ldots, b-1, b, b+1, \ldots$ can be represented by: $\{\ \mathrm{b} = a+5 = c+12 = d+8\}$

Integer Offsets (contd.)

- Can assume input equations of the form $a = b + k$ or of the form $\cdot(a, b + k_b) = c + k_c$ (not hard to see)
- Pending now contains eqs like $a = b{+}k$
- Representative(a) returns pair (b, k) such that $b = a{+}k$
- Similarly for Class lists, Lookup table, and Use lists.
- Obtain algorithm with same complexity!

BUT

If also atoms $s > t$ are allowed in (positive conjunction) input then satisfiability becomes NP-hard (reduce k-coloring, see paper for details).

CC: our algorithm with offsets

While $Pending \neq \emptyset$ Do
  remove $a = b + k$ with representative $a' = b' + k_{b'}$ from $Pending$
  If $a' \neq b'$ and, wlog., $|ClassList(a')| \leq |ClassList(b')|$ Then
    For each $c + k_c$ in $ClassList(a')$ Do
      set $rep(c)$ to $(b', k_c - k_{b'})$ and add it to $ClassList(b')$
    EndFor
    For each $\cdot(c, d + k_d) = e + k_e$ in $UseList(a')$ Do
      If $Lookup(c', r(d + k_d))$ is $f + k_f$ and $r(f + k_f) \neq r(e + k_e)$ Then
        add $e = f + (k_f - k_e)$ to $Pending$
      EndIf
      set $Lookup(c', r(d + k_d)$ to $r(e + k_e)$
      add $\cdot(c, d + k_d) = e + k_e$ to $UseList(b')$
    EndFor
  ElseIf $a' = b'$ and $k_{b'} \neq 0$ Then return *unsatisfiable*
  EndIf
EndWhile

Final remarks

- Simple but efficient algorithm, also for integer offsets
- $DPLL(=)$ needs CC with backtracking and also needs dealing with negative equalities
- First implementation of $DPLL(=)$ finished
- Results encouraging
- Still working on learning and heuristics