On the Use of Extended Resolution in Propositional Reasoning

Carsten Sinz

Institute for Formal Models and Verification
Johannes Kepler University Linz
Linz, Austria

January 2006

Overview

- Motivation
- Proof Generation in SAT-Solvers
- Extended Resolution for Compressing Proofs
- Extended Resolution for BDD Constructions
- Conclusion

Part I:
Motivation

Background

- Propositional logic used in many real-world applications today:
  - Hardware & software verification, planning, FPGA routing, product configuration
- Efficient decision procedures available
  - SAT-Solvers can handle instances with 100,000 variables and millions of clauses
- Justification of results needed
  - To locate errors, for debugging, ...

Example: Product Configuration

Options available for Mercedes-Benz's C class: (excerpt, total: 692)

231 garage door opener integrated into interior mirror
280 steering wheel in leather design (two-colored) with chrome clip
550 trailer appliance
581 comfort air-conditioning THERMOTRONIC
671 light metal wheels 4x, 7 spoke design
673 high-capacity battery
772 AMG (sports) styling

Restrictions for Mercedes-Benz's C class: (excerpt, total: 952)

AMG styling (772) cannot be combined with trailer appliance (550).

Comfort air-conditioning (581) requires high-capacity battery (673), except when combined with gasoline engines with 2.6 or 3.2 liter cylinder capacity.

General Setting

- Propositional logic SAT problem
  - Formulae in CNF: $F = C_1 \wedge \ldots \wedge C_m$
    with $C_i = l_{i,1} \vee \ldots \vee l_{i,k_i}$ and $l_{i,j} \in \{x, \neg x \mid x \in V\}$
  - Question: Is there an assignment to the variables in *V* such that *F* evaluates to true?
- If yes, a model is found
  - Delivers information to the user (solution, counter-example)
  - Can easily be checked for correctness
- If no, no model is found
  - No additional information for the user
  - Can we trust the SAT-Solver program? Is there really no model? What kind of „certificate“ can we obtain?

SAT Applications: Interpretation of Unsatisfiable Instances

- FPGA routing: channel unroutable
  - What is the reason for this? Where is the “hot spot”?
- Planning: no plan with the given restrictions
  - Which restrictions could be changed?
- Product configuration: no product instance with the given specification
  - How should the specification be changed?
- Finite mathematics (e.g. Quasigroup existence problems): no structure of a given size
  - Why is this the case? (resp.: Is there really no structure of this size or is the SAT-Solver faulty?)

Solution

- Generate proofs!
  - (Refutation) Proofs serve as certificates for unsatisfiability
  - Can be checked easily (polynomial in the length of the proof)
  - Resolution-based SAT-Solvers can generate proofs „as a by-product“
- Many SAT-Solvers resolution based
  - Thus easily extendable to generate resolution proofs

Part II:
Proof Generation in SAT-Solvers

SAT-Solvers

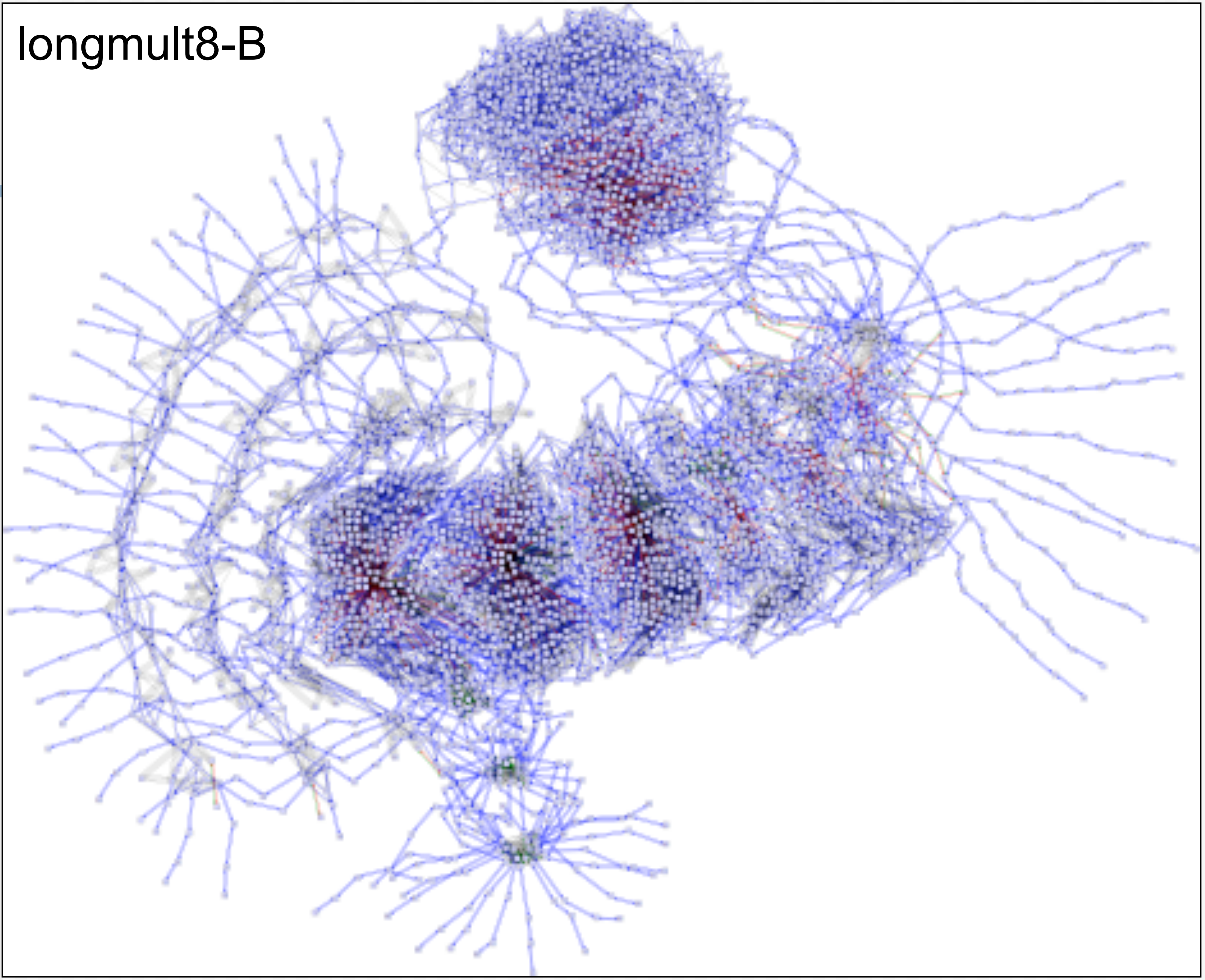
- Predominant algorithm : DPLL (Davis-Putnam-Logemanm-Loveland)

```
boolean DPLL(ClauseSet S)
{
    while (S contains a unit clause {L}) {          // unit propagation
        delete from S all clauses containing L;     // u. subsumption
        delete ¬L from all clauses in S;            // u. resolution
    }
    if (∅ ∈ S) return false;
    if (S = ∅) return true;
    choose a literal L occurring in S;
    if (DP(S ∪ {{L}}) return true;
    else return DP(S ∪ {{¬L}});
}
```

SAT-Solvers: Recent Extensions

- Recent (influential) enhancements:
  - Clause (no-good) learning [MarquesSilva&Sakallah 1996]
  - Fast (lazy) Boolean constraint propagation (*watched literals* data structure) [Moskewicz *et al*. 2001]
  - Improved (dynamic) variable selection heuristics (VSIDS, locality considered) [Moskewicz *et al*. 2001]
  - Rapid random restarts (to overcome heavy-tail behavior) [Gomes *et al*. 1998]
  - Clause set compression (deletion of subsumed clauses) [Biere 2004]
- Also important: instances occurring in practice are highly structured

longmult8-B

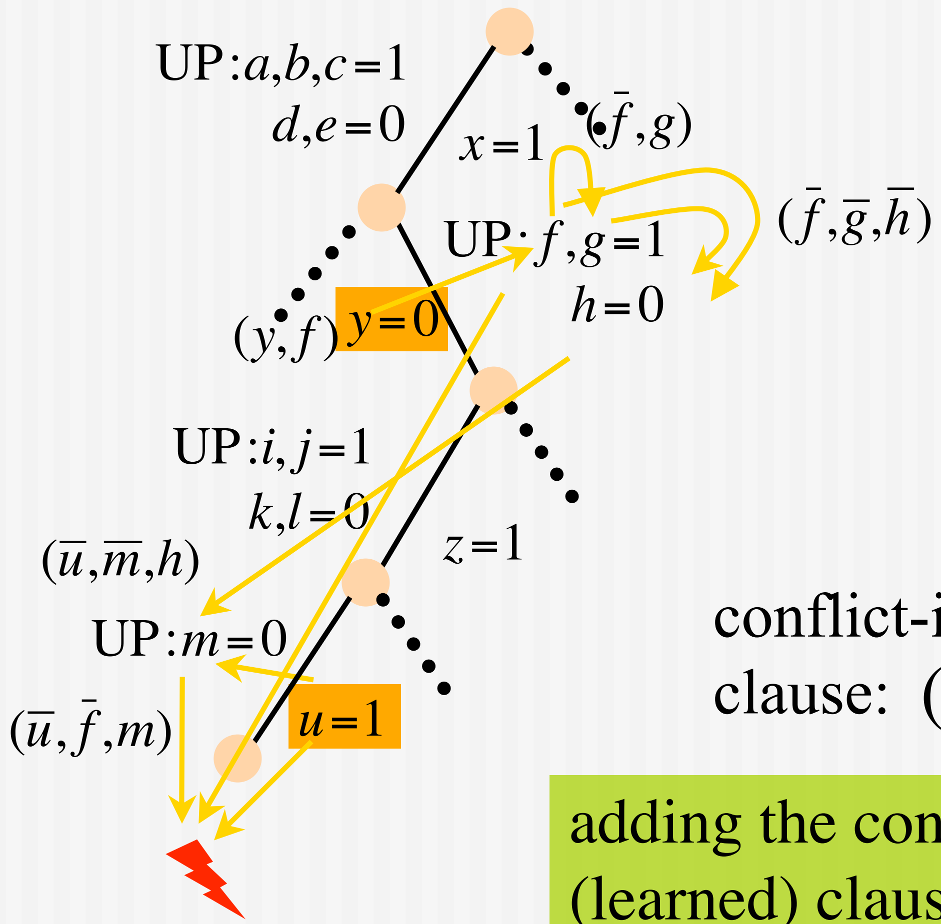


Enhanced DPLL Algorithm with Learning

```
boolean DPLL-Enhanced
{
    forever {
            ok = propagate_units();
            if (!ok) {              // conflicting assignment
                    generate_and_add_conflict_clause();
                    new_level = backtrack();
                    if (new_level < 0) return false;
            }
            if no more open variables return true;
            decide();           // assign value to open literal
    }
}
```

Lemma Generation: Example

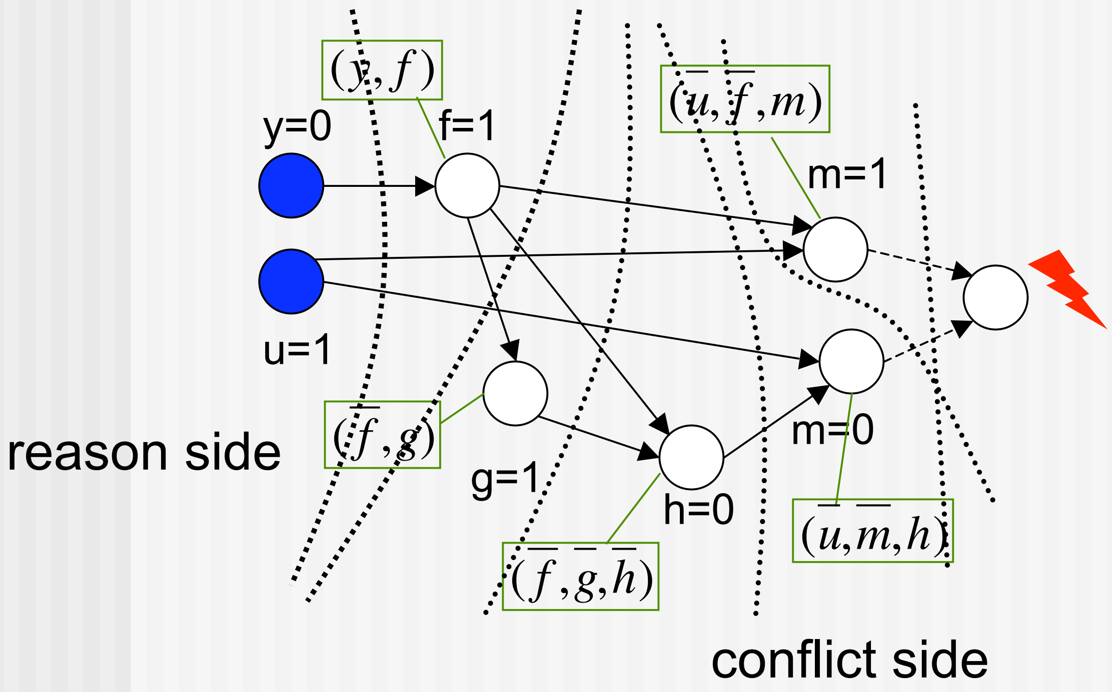
$(\bar{u},\bar{m},h)$
$(\bar{u},\bar{f},m)$
$(\bar{f},\bar{g},\bar{h})$
(y,f)
$(\bar{f},g)$
$\vdots$
$(\bar{x},a)$
$(\bar{x},b,\bar{a})$
$(\bar{x},\bar{a},\bar{b},c)$
$(\bar{a},\bar{d})$
$(\bar{b},d,\bar{e})$
$(\bar{c},d,e)$
$\vdots$



conflict-induced clause: $(y,\bar{u})$

adding the conflict-induced (learned) clause avoids repeated search

Resolution Proofs for Generated Lemmas



resolution proof:

$(\bar{u}, \bar{f}, m)$ $\quad$ $(\bar{u}, \bar{m}, h)$

$(\bar{u}, \bar{f}, h)$ $\quad$ $(\bar{f}, \bar{g}, \bar{h})$

$(\bar{u}, \bar{f}, \bar{g})$ $\quad$ $(\bar{f}, g)$

$(\bar{u}, \bar{f})$ $\quad$ (y, f)

$(\bar{u}, y)$

ordering: (y, f) $(\bar{f}, g)$ $(\bar{f}, \bar{g}, \bar{h})$ $(\bar{u}, \bar{m}, h)$ $(\bar{u}, \bar{f}, m)$

Proofs in the DPLL Algorithm with Learning

- Resolution proofs for lemmas are *trivial*
  - input (i.e. also linear)
  - regular (i.e. all resolution variables are distinct)

  (Notion defined in [Beame *et al*. 2003])
- Resolution refutation for *F* is generated by
  - Taking proofs of all lemmas used to derive the empty clause
- Proof of a lemma (resp. involved input clauses) also called a *proof chain*

State of the Art

- Proof („trace“) generation built into some SAT-Solvers
  - Chaff, MiniSAT, booleforce
- Proofs may become large!
  - 929 MB for a proof trace of PHP_{11} with booleforce
- Core extraction can alleviate situation

Core Extraction

- Idea: determine a smallest possible clause set that is still unsatisfiable
  - MUS (minimal unsatisfiable subformula)
    - Approximation algorithm [Bruni&Sassano 2000]
    - Based on iteratedly solving modified SAT instances [Oh *et al*. 2004]
  - Core extraction
    - Based on resolution of learned clauses [Zhang&Malik 2003]
    - Core contains clauses in the lemma's proof chains
    - May be applied iteratively
    - Also implemented in booleforce

Challenging Problems

- How can smaller proofs be obtained?
- Proofs for non-resolution decision procedures (e.g. Binary Decision Diagrams)

Idea: Use stronger proof systems to represent proofs

Part III:
Extended Resolution for Compressing Proofs

Extended Resolution (ER)

- Resolution Rule + Extension Rule
- Resolution Rule:

$$\frac{C \dot\cup \{l\} \qquad D \dot\cup \{\bar{l}\}}{C \cup D}$$

- Extension Rule:
  - Add clauses for definition $x \leftrightarrow F$
    - x new variable (i.e. not occurring in original formula or previous definitions)
    - F arbitrary formula (original paper: only $F = l_1 \wedge l_2$ allowed)
  - First proposed by Tseitin in 1970

General Ideas of Proof Compression

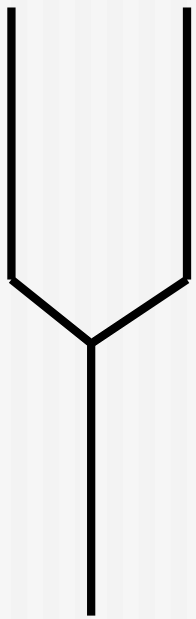
- Merge proof chains
- Exploit symmetries

- Related ideas proposed in the context of first-order logic:
  - Dynamically add definitions [Eder 1990]
  - Quantifier introduction [Egly 1992]
  - Function introduction [Baaz&Leitsch 1992; Egly 1993]
  - Substitution formulae (δ^m-resolution) [Peltier 2005]

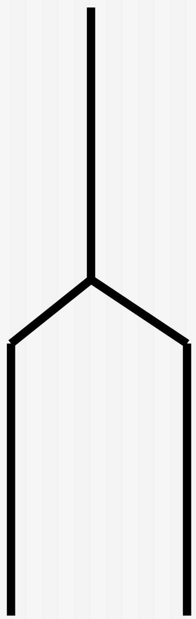
Merging Proof Chains

- Observation: Many proof chains differ by only a few clauses, e.g.
  - (-28 -22)(25 26 27 28 29 30)(-11 -29)(5 11)(26 25 -5 27)(-25 -31)(-15 -27)(-26 -38)(37 38 39)(-37 -31)(-15 -39)(-6 -30)(5 4 6)(25 27 26 -4 29)(24 22)(-6 -24) to prove (-15 -31), and
  - (-28 -22)(25 26 27 28 29 30)(-11 -29)(5 11)(26 25 -5 27)(-26 -32)(-15 -27)(-25 -37)(37 38 39)(-38 -32)(-15 -39)(-6 -30)(5 4 6)(25 27 26 -4 29)(24 22)(-6 -24) to prove (-15 -32)
  - are proof lanes in the proof of PHP_6
- Idea: Re-order proof steps and merge common parts of proof chains

Merging Proof Chains: Constellations



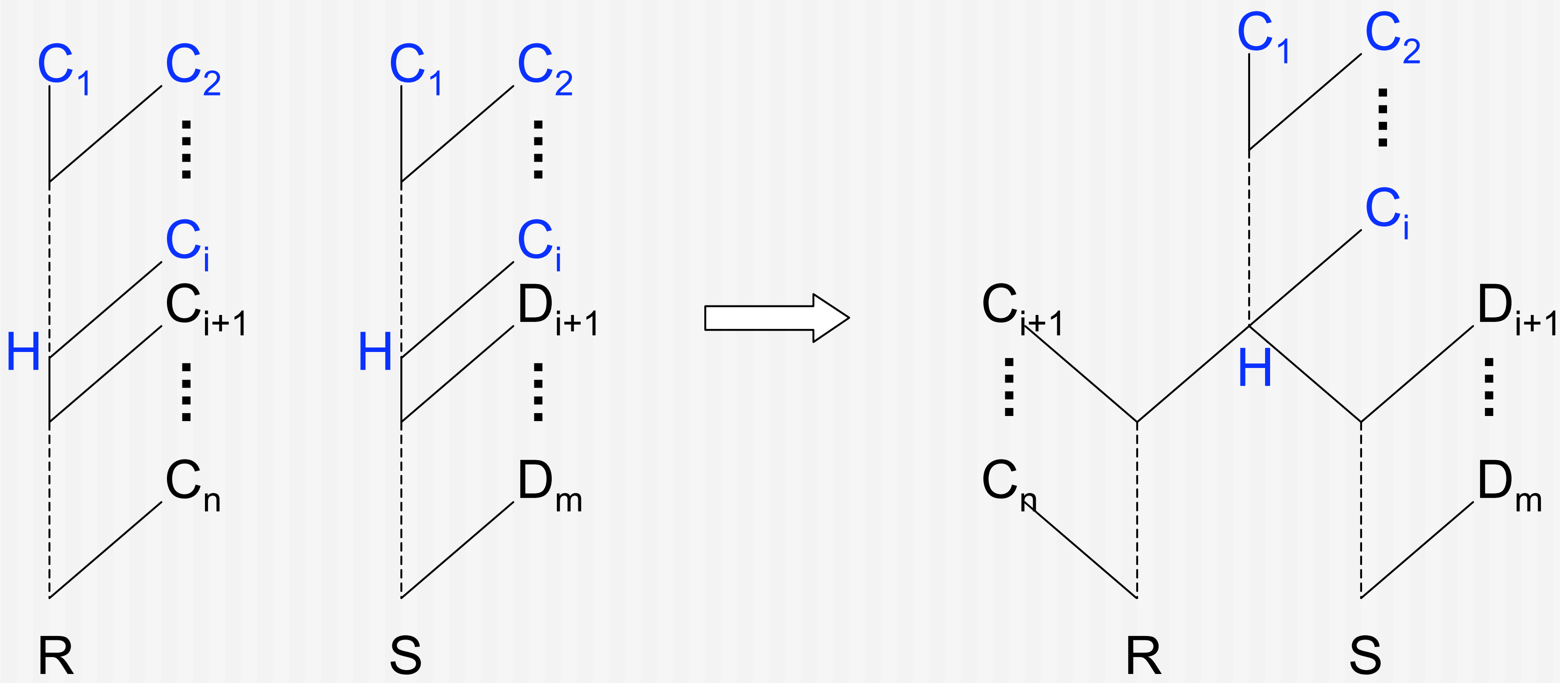
common postfix



common prefix

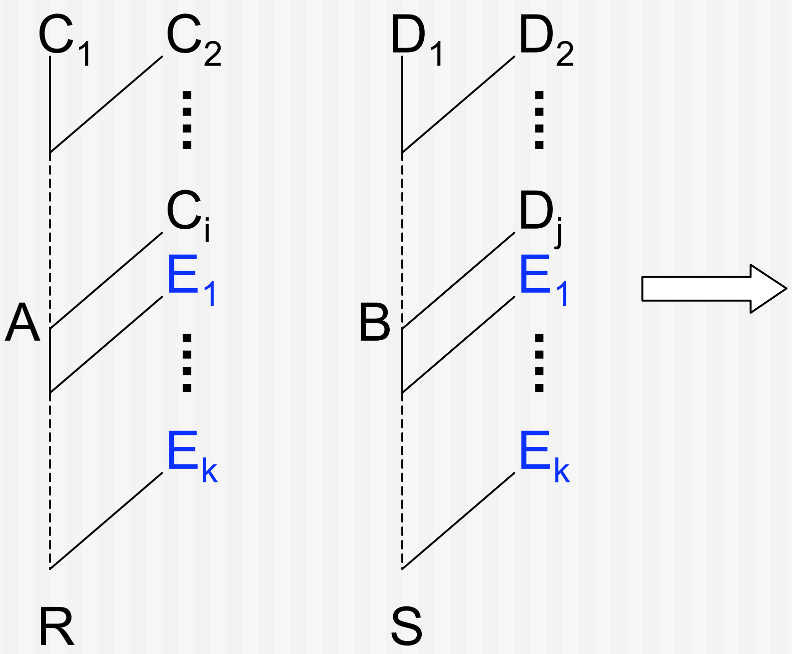
mixed cases also possible

Merging Proof Chains: Common Prefix



merging easily possible: equivalent to three shorter chains (no ER required)

Merging Proof Chains: Common Postfix



Assume $A = (a_1,\dots,a_p,c_1,\dots,c_r)$
and $B = (b_1,\dots,b_q,c_1,\dots,c_r)$
(with c_i common literals of A,B; none of the a_i or b_i must be resolved below A,B)

Define $w_1 \leftrightarrow (a_1 \vee \dots \vee a_p)$
and $w_2 \leftrightarrow (b_1 \vee \dots \vee b_q)$
and $w \leftrightarrow w_1 \wedge w_2$

?

Further, let $AB^* = (w,c_1,\dots,c_r)$.

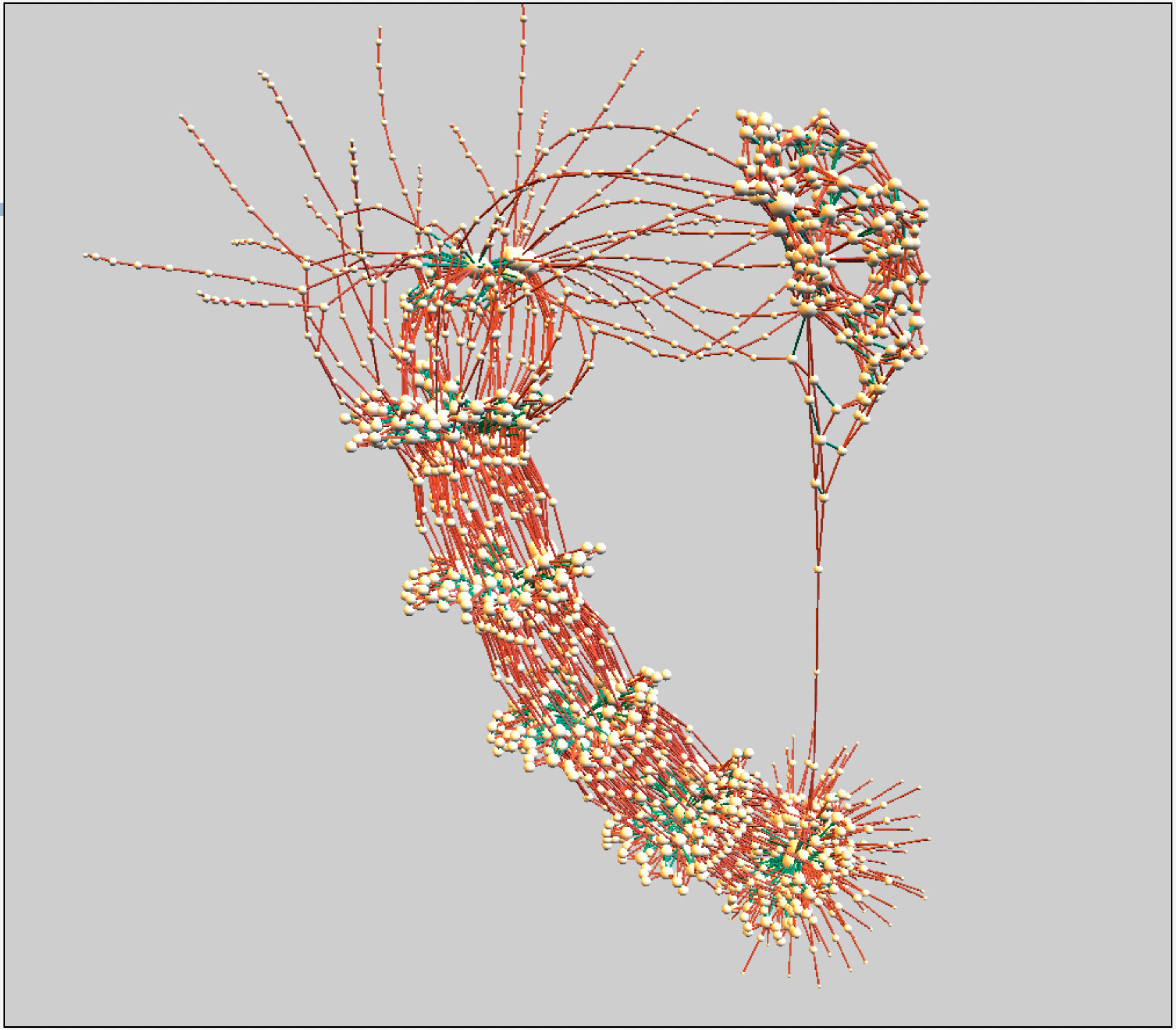
Then $A,B \vdash_{ER} AB^*$
and $AB^*,E_1,\dots,E_k \vdash_{ER} RS^* = R[a_1,\dots,a_p/w]$ $(= S[b_1,\dots,b_q/w])$
(as none of the a_i or b_i is resolved below A,B)
and $RS^* \vdash_{ER} R, S$

merging not easily possible: ER required

We thus have a (possibly shorter) ER proof for R and S.

Exploiting Symmetries

- Assume F symmetric, i.e. $F = \pi(F)$ for some permutation π of the literals.
- Idea:
  - Instead of many proofs for symmetric clauses C, $\pi(C)$, π^2(C)..., derive C's symmetric closure
    $$\mathrm{SymCl}(C) = C \wedge \pi(C) \wedge \pi^2(C) \wedge \pi^3(C) \wedge \ldots$$
    with one ER proof
  - Advantage: only one proof chain for C, $\pi(C)$, $\pi^2(C)$, ...
- Work in progress



Part IV:

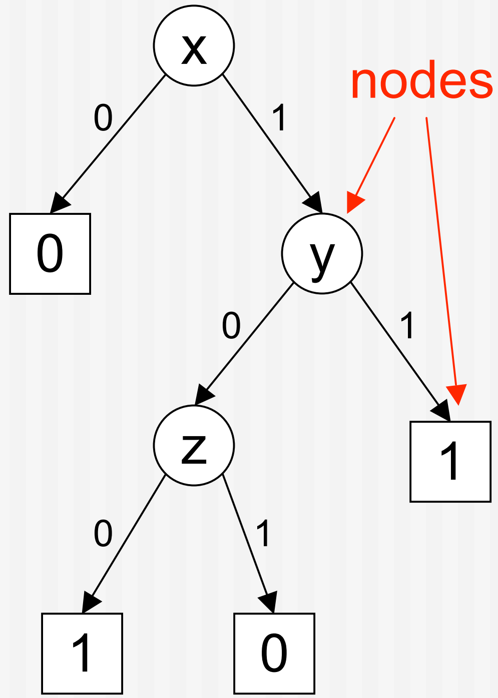
Extended Resolution Proofs out of BDD Computations

Binary Decision Diagrams (BDDs)

- BDDs: Graph-based data structure to represent Boolean functions
  - Based on Shannon expansion:

    $$f = (x \rightarrow f|_{x=1}) \wedge (\neg x \rightarrow f|_{x=0})$$

  - Isomorphic sub-graphs shared
  - Canonical representation when variable order is fixed



BDD representing formula $x \wedge (y \vee \neg z)$

BDDs (cont´d)

- Common in HW verification
- BDDs typically built bottom-up from smaller ones using BDD constructors (`BDD_and`, `BDD_or`,...)
- BDDs as SAT-Solver (for $S = \{C_1,\ldots,C_m\}$):
  1. Convert clauses C_i to BDDs c_i
  2. Using `BDD_and`, construct BDDs h_i for partial conjunctions $C_1 \wedge \ldots \wedge C_i$
  3. S is unsatisfiable iff $h_m=0$

Proofs from BDD Constructions

- Algorithm:
  1. Generate BDDs for clauses c_i and partial conjunctions h_i as indicated
  2. Add definitions for all used nodes: $f \leftrightarrow ITE(x,f_0,f_1)$
  3. Generate ER proofs for
     a) $S \vdash_{ER} c_i$
     b) $S \vdash_{ER} c_1 \wedge c_2 \rightarrow h_2$, $\quad S \vdash_{ER} h_{i-1} \wedge c_i \rightarrow h_i$
     c) $S \vdash_{ER} h_m$
- Parts *a)* and *c)* easy, *b)* by recursion
- Details in [Sinz&Biere 2006]: first experimental results promising (trace size reduced from 929 MB to 8 MB for PHP_{11})

Proofs from BDD Constructions Recursive Step

$$
\dfrac{
\dfrac{(h x \bar{h}_0) \qquad \dfrac{(\bar{g} x g_0) \qquad \dfrac{(\bar{f} x f_0) \qquad \dfrac{\vdots}{(\bar{f}_0 \bar{g}_0 h_0)}}{(\bar{f} x \bar{g}_0 h_0)}}{(\bar{f} \bar{g} x h_0)}}{(\bar{f} \bar{g} h x)}
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{(\bar{f}_1 \bar{g}_1 h_1)} \qquad (\bar{f} \bar{x} f_1)}{(\bar{f} \bar{x} \bar{g}_1 h_1)} \qquad (\bar{g} \bar{x} g_1)}{(\bar{f} \bar{g} \bar{x} h_1)} \qquad (h \bar{x} \bar{h}_1)}{(\bar{f} \bar{g} h \bar{x})}
}{(\bar{f} \bar{g} h)}
$$

With node definitions:

$$
\begin{array}{lll}
f \leftrightarrow x \,?\, f_1 : f_0 & \qquad & (\bar{f} \bar{x} f_1)(\bar{f} x f_0)(f \bar{x} \bar{f}_1)(f x \bar{f}_0) \\
g \leftrightarrow x \,?\, g_1 : g_0 & \qquad & (\bar{g} \bar{x} g_1)(\bar{g} x g_0)(g \bar{x} \bar{g}_1)(g x \bar{g}_0) \\
h \leftrightarrow x \,?\, h_1 : h_0 & \qquad & (\bar{h} \bar{x} h_1)(\bar{h} x h_0)(h \bar{x} \bar{h}_1)(h x \bar{h}_0)
\end{array}
$$

Conclusion

- Shown two applications of Ext. Resolution:
    1. Proof compression for DPLL-based SAT-Solvers
    2. ER-Proofs from BDD constructions
- Applications in
    - HW & SW verification, configuration, ...
    - “Certificate” generation for SAT-Solvers
- Outlook
    - Extension to QBF
    - Symmetry