Extracting Unsatisfiable Cores for LTL via Temporal Resolution

Viktor Schuppan

TIME 2013, Pensacola, FL, USA, September 26-28, 2013

LTL as a Specification Language

LTL + relatives widely used specification languages; methodologies exist:

- Embedded systems: e.g., [EF06]; [Pil+06].
- Business processes: e.g., [PA06]; [Awa+12].

But:

Beer et al. (IBM) [Bee+01]:

> [...] during the first formal verification runs of a new hardware design, typically 20 % of formulas are found to be trivially valid, and that trivial validity always points to a real problem in either the design or its specification or environment.

Bloem et al. [Blo+07] in a work on LTL synthesis:

> [...] writing a complete formal specification [...] was not trivial.

> Although this approach removes the need for verification [...] the specification itself still needs to be validated.

Efficient working with LTL requires effective debugging techniques.

LTL Specification Validation with Satisfiability

Examples of satisfiability in validation checks of an LTL specification ϕ:

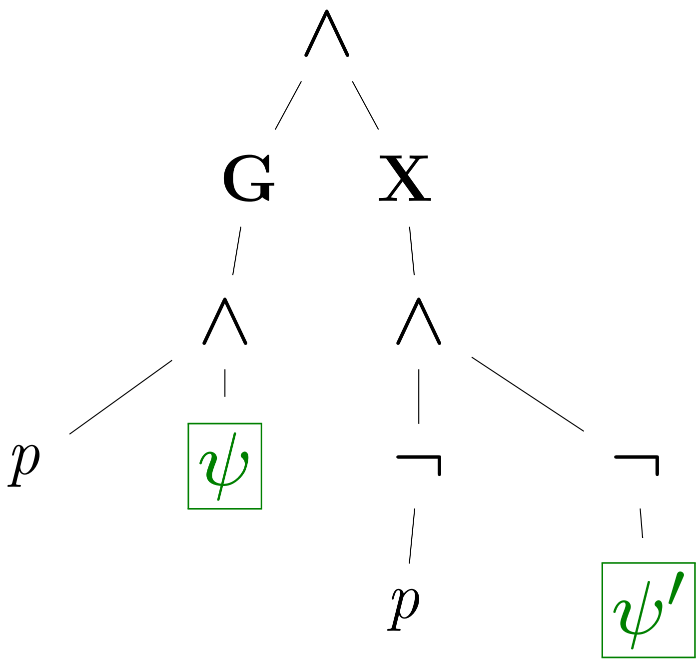
- Satisfiability of ϕ (e.g., [RV10,Awa+12]).
- Feasibility of LTL scenario ϕ' in ϕ: satisfiability of $\phi \wedge \phi'$ (e.g., [Pil+06]).
- Implication of desired LTL property ϕ'' by ϕ: unsatisfiability of $\phi \wedge \neg\phi''$ (e.g., [Pil+06]).

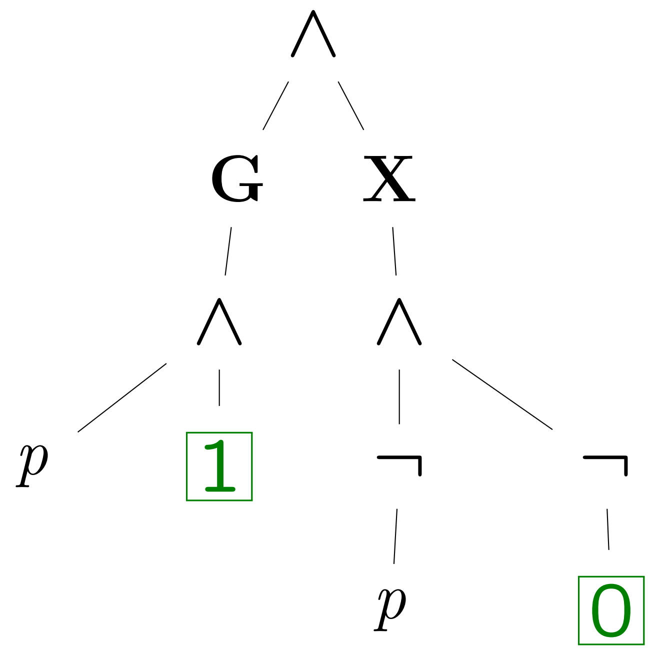
An unsatisfiable core (UC) is an unsatisfiable formula ϕ' that is derived from another unsatisfiable formula ϕ. ϕ' focuses on a reason for ϕ being unsatisfiable.

UCs can help understanding results of validation checks.

Failure-inducing input minimization (e.g., [ZH02]) is established in many domains, e.g., linear programming (e.g., [CD91]), constraint satisfaction (e.g., [Bak+93]), compilers (e.g., [Wha94]), SAT (e.g., [BS01]), declarative specifications (e.g., [Shl+03]), and LTL satisfiability (e.g., [Sch12]) and realizability (e.g., [Cim+08]).

UCs via Syntax Trees





$$(\mathbf{G}(p \wedge \boxed{\psi})) \wedge (\mathbf{X}(\neg p \wedge \neg \boxed{\psi'})) \qquad (\mathbf{G}(p \wedge \boxed{1})) \wedge (\mathbf{X}(\neg p \wedge \neg \boxed{0}))$$

Replace some positive polarity occurrences of subformulas with 1 and some negative polarity occurrences of subformulas with 0 while preserving unsatisfiability ([Sch12,KV03]).

Temporal Resolution (TR) as a Basis for Extracting UCs

Deletion-based extraction of UCs (e.g., [MS10]) is straightforward using any solver but may be expensive.

Resolution-based extraction of UCs

- Common, e.g., in SAT [VG02].
- Resolution method for LTL suggested by Fisher [Fis91,FDP01] and implemented in `TRP++` [HK03,HK04,trp++]; sources available.
- `TRP++` competitive in experimental evaluation [SD11]; in particular also on unsatisfiable instances.
- Access to and reasoning about proof is straightforward.
- BDD-based `NuSMV` [Cim+02] also performed well on unsatisfiable instances; but: BDD layer as complication.
- Tableau-based solvers `LWB` [Heu+95] and `pltl` [pltl] also provide good access to proof; but: didn't do well on unsatisfiable instances.

Contents

Separated Normal Form (SNF)

TR works on a clausal normal form called Separated Normal Form (SNF) [FDP01].

Let $p_1, \ldots, p_n$, $q_1, \ldots, q_{n'}$, l with $0 \leq n, n'$ be literals such that $p_1, \ldots, p_n$ and $q_1, \ldots, q_{n'}$ are pairwise different.

$(p_1 \vee \ldots \vee p_n)$ is an initial clause.

$(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{X}(q_1 \vee \ldots \vee q_{n'}))))$ is a global clause.

$(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{F}(l))))$ is an eventuality clause.

$()$ or $(\mathbf{G}())$, denoted $\Box$, stand for 0 or $\mathbf{G}(0)$ and are called empty clause.

Let $c_1, \ldots, c_n$ with $0 \leq n$ be SNF clauses. Then $\bigwedge_{1 \leq i \leq n} c_i$ is an LTL formula in SNF.

There exists a structure-preserving translation from an LTL formula into an equisatisfiable formula in SNF [FDP01].

Initial and Step Resolution

Initial and step resolution are straightforward extensions of propositional resolution.

They differentiate between initial, global current, and global next literals to allow resolution between 2 clauses each of which may be initial or global.

Example 1, initial and global clause:

$$\frac{(P \vee l) \qquad (\mathbf{G}((\neg l) \vee Q))}{(P \vee Q)}$$

Example 2, 2 global clauses:

$$\frac{(\mathbf{G}(P \vee l)) \qquad (\mathbf{G}((Q) \vee (\mathbf{X}((\neg l) \vee R))))}{(\mathbf{G}((Q) \vee (\mathbf{X}(P \vee R))))}$$

Eventuality Resolution

Goal

$$\frac{(\mathbf{G}(P \vee \mathbf{F}l)) \qquad (\mathbf{G}(Q \vee \mathbf{XG}\neg l))}{(\mathbf{G}(P \vee Q \vee l))}$$

Loop Search for l

Let $Q \equiv 0$.

Perform loop search iterations until done.

Loop Search Iteration for l

Assume all global clauses with non-empty $\mathbf{X}$ part.

Assume all global clauses with empty $\mathbf{X}$ part, shifted 1 step into the future.

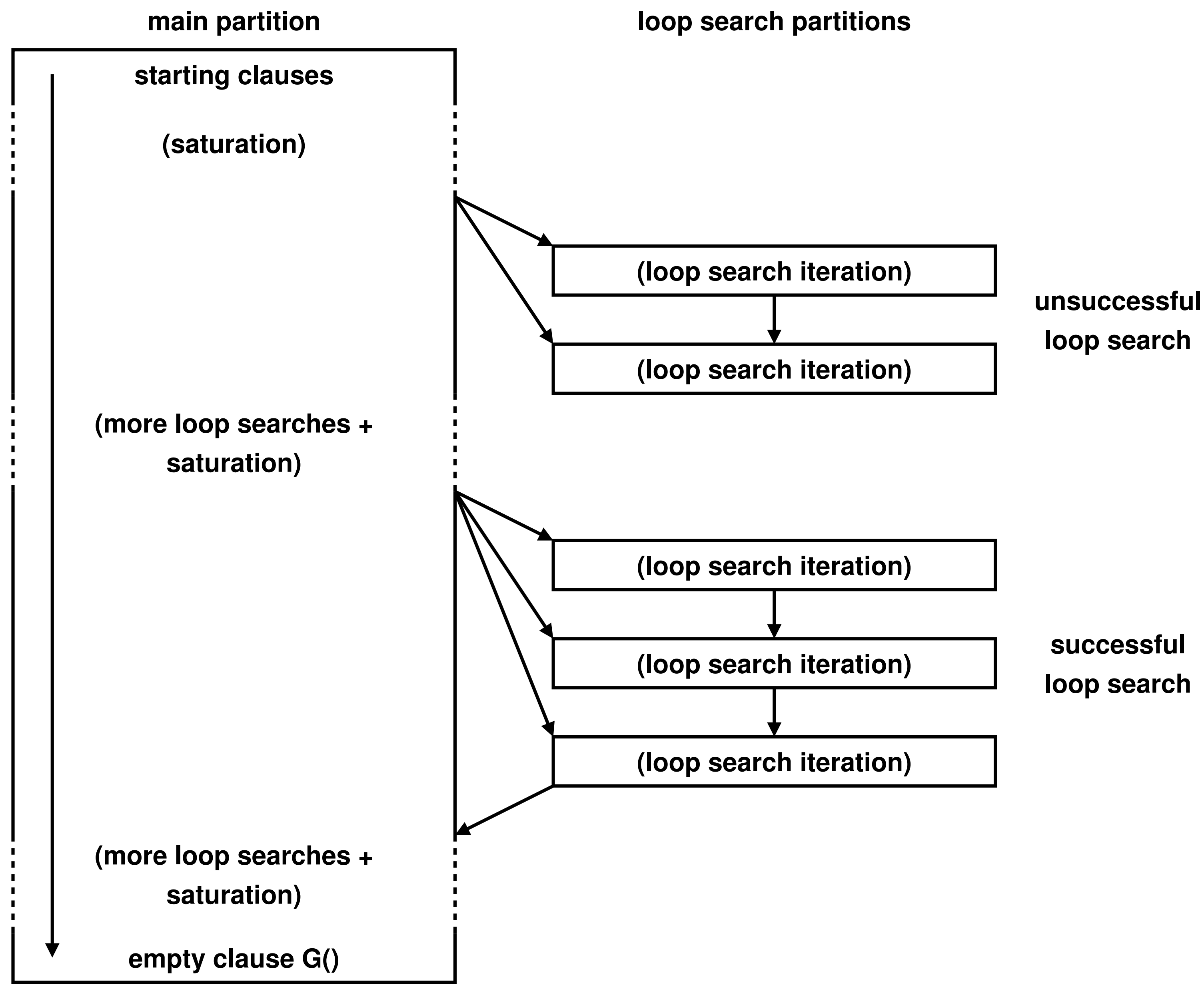
Assume $(\mathbf{GX}(Q \vee l))$.

Deduce, using step resolution between clauses with non-empty $\mathbf{X}$ part, R.

Distinguish 3 cases:

- $R \leq Q$: done, found Q as desired.
- $Q < R < 1$: perform next iteration with $Q \equiv R$.
- $R = 1$: done, no Q found at this point.

Scheduling and Flow of Information



Extraction of a UC with a Resolution Graph

During the execution of the TR algorithm construct a resolution graph.

- Clauses are vertices.
- Applications of production rules induce edges from premises to conclusions.

If the empty clause has been derived

- Construct the set of clauses backward reachable from the empty clause.
- Intersect with set of starting clauses to obtain a UC in SNF.

So far, so trivial. Some optimizations follow.

Resolution graph interesting in its own right as a proof object that enables to extract further useful information. See outlook.

Set of Premises to Include in Resolution Graph

1. Several production rules have an eventuality clause as a premise. In three cases there need not be an edge from that premise to the conclusion as that eventuality clause will be included in the resolution graph via other edges.

2. A successful loop search finds Q and proves that it is a fixed point. Only the proof of Q being a fixed point is required in the resolution graph — which happens in the last iteration of a successful loop search. Previous iterations only serve to derive Q and can be discarded (no edges from one loop search iteration to the next).

Minimality of Set of Premises to Include in Res. Graph

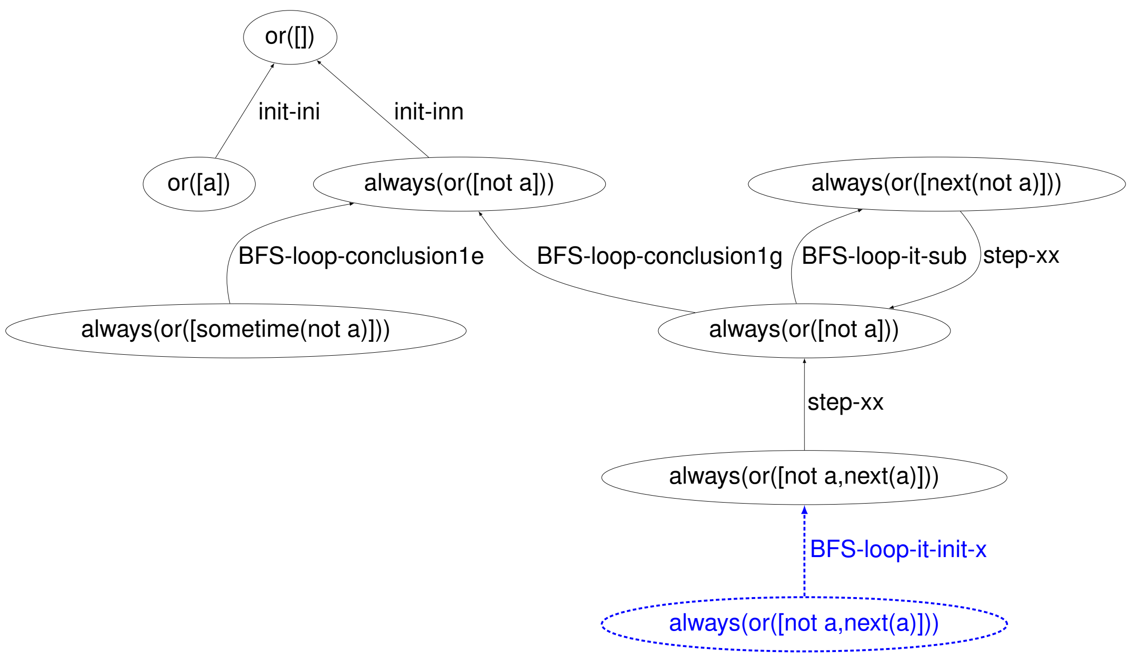
To show that some premise of some production rule is needed to obtain a UC, find

– a minimal UC in SNF C^{uc},

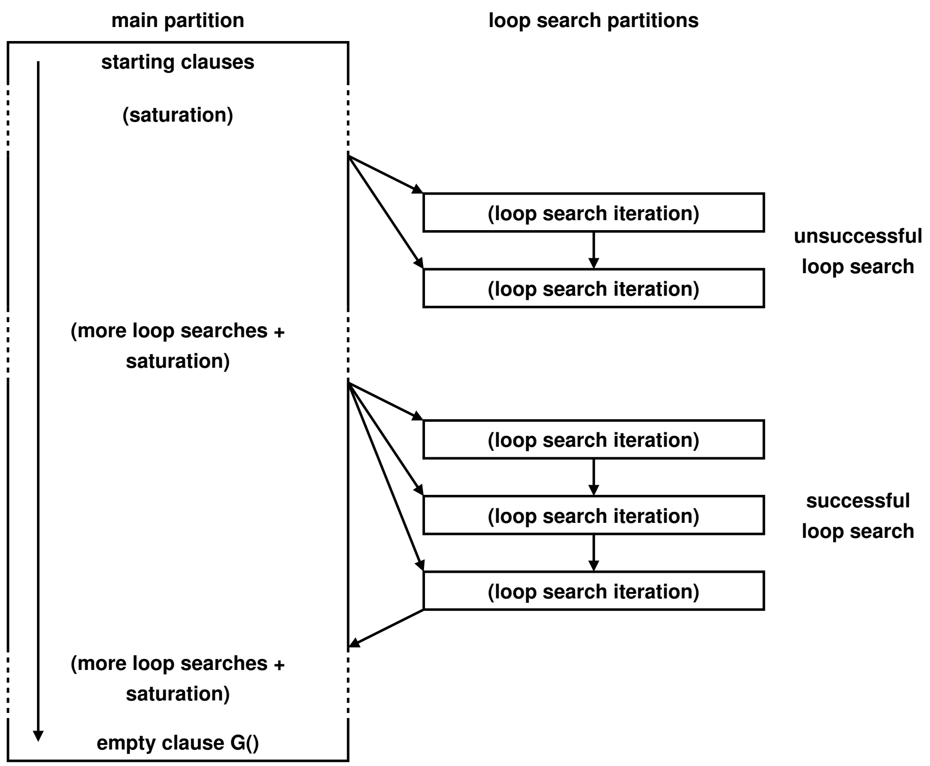
– such that in the backward reachable part of its resolution graph,

– some clause in C^{uc} is backward reachable from the empty clause only via an edge representing that premise in that production rule.

Example: $\{(a), (\mathbf{G}((\neg a) \vee (\mathbf{X}(a)))), (\mathbf{G}(\mathbf{F}(\neg a)))\}$



Pruning the Resolution Graph



1. After completion of a loop search there will be no further edges from those loop search partitions to main partition. Prune vertices not backward reachable from the main partition.

2. With earlier optimization a failed loop search iteration has no outgoing edges. Prune failed loop search iteration right away.

From LTL to SNF and Back

Structure preserving translation (e.g., [PG86]) from LTL to SNF.

LTL $(\mathbf{G}p) \wedge (\mathbf{X}((\neg p) \wedge (q \vee r)))$

SNF, UC in SNF

$$
\begin{array}{l}
\{x_\phi, \\
\ (\mathbf{G}(x_\phi \rightarrow x_{\mathbf{G}p})), \\
\ (\mathbf{G}(x_\phi \rightarrow x_{\mathbf{X}((\neg p)\wedge(q\vee r))})), \\
\ (\mathbf{G}(x_{\mathbf{G}p} \rightarrow p)), \\
\ (\mathbf{G}(x_{\mathbf{G}p} \rightarrow \mathbf{X}x_{\mathbf{G}p})), \\
\ (\mathbf{G}(x_{\mathbf{X}((\neg p)\wedge(q\vee r))} \rightarrow \mathbf{X}x_{(\neg p)\wedge(q\vee r)})), \\
\ (\mathbf{G}(x_{(\neg p)\wedge(q\vee r)} \rightarrow x_{\neg p})), \\
\ (\mathbf{G}(x_{(\neg p)\wedge(q\vee r)} \rightarrow x_{q\vee r})), \\
\ (\mathbf{G}(x_{\neg p} \rightarrow \neg p)), \\
\ (\mathbf{G}(x_{q\vee r} \rightarrow q \vee r))\}
\end{array}
$$

UC in LTL $(\mathbf{G}p) \wedge (\mathbf{X}((\neg p) \wedge 1))$

$q \vee r$ does not appear on any right hand side of an implication of a clause in the UC in SNF; it is therefore replaced with 1 in the UC in LTL.

Minimal UCs 1

A UC ϕ^{uc} in LTL is minimal iff no positive polarity occurrence of a subformula of ϕ^{uc} can be replaced with 1 and no negative polarity occurrence of a subformula of ϕ^{uc} can be replaced with 0 without making ϕ^{uc} satisfiable.

UCs obtained so far may not be minimal.

Perform deletion-based minimization (e.g., [MS10]).

May be expensive in general, but can do it on already reduced formula.

Note: minimization must be performed on LTL rather than SNF levels.

Example for non-minimality in LTL if minimization is performed on SNF level:

LTL (= UC in LTL)

$$(\neg p) \wedge ((\mathbf{G}\neg q) \wedge (p\mathbf{U}q))$$

SNF, a minimal UC in SNF

$$
\begin{aligned}
\{&x_\phi,\\
&(\mathbf{G}(x_\phi \rightarrow x_{\neg p})),\\
&(\mathbf{G}(x_{\neg p} \rightarrow \neg p)),\\
&(\mathbf{G}(x_\phi \rightarrow x_{(\mathbf{G}\neg q)\wedge(p\mathbf{U}q)})),\\
&(\mathbf{G}(x_{(\mathbf{G}\neg q)\wedge(p\mathbf{U}q)} \rightarrow x_{\mathbf{G}\neg q})),\\
&(\mathbf{G}(x_{\mathbf{G}\neg q} \rightarrow \mathbf{X}x_{\mathbf{G}\neg q})),\\
&(\mathbf{G}(x_{\mathbf{G}\neg q} \rightarrow x_{\neg q})),\\
&(\mathbf{G}(x_{\neg q} \rightarrow \neg q)),\\
&(\mathbf{G}(x_{(\mathbf{G}\neg q)\wedge(p\mathbf{U}q)} \rightarrow x_{p\mathbf{U}q})),\\
&(\mathbf{G}(x_{p\mathbf{U}q} \rightarrow (q \vee p))),\\
&(\mathbf{G}(x_{p\mathbf{U}q} \rightarrow (q \vee \mathbf{X}x_{x_{p\mathbf{U}q}}))),\\
&(\mathbf{G}(x_{p\mathbf{U}q} \rightarrow \mathbf{F}q))\}
\end{aligned}
$$

Implementation, Experimental Setup

Implementation

– on top of `TRP++` [HK03,HK04,trp++]

– data structures: C++ standard library [SL95,Jos12]

– graph operations: Boost Graph Library [bgl,SLL02]

Experimental Setup

– Intel Core i7 M 620 @ 2 GHz

– Ubuntu 12.04

– time limit: 600 seconds

– memory limit: 6 GB

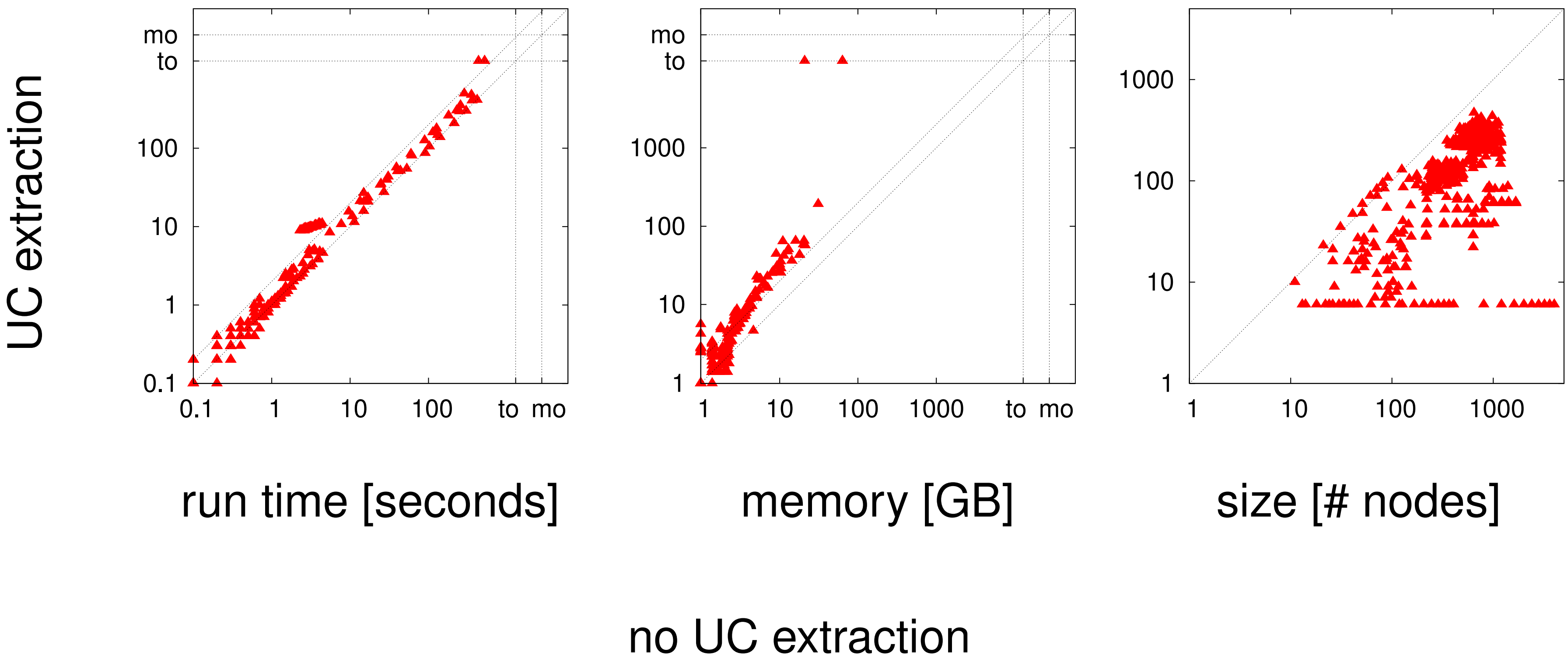
– time and memory measured and bounded with `run` [run]

Benchmarks

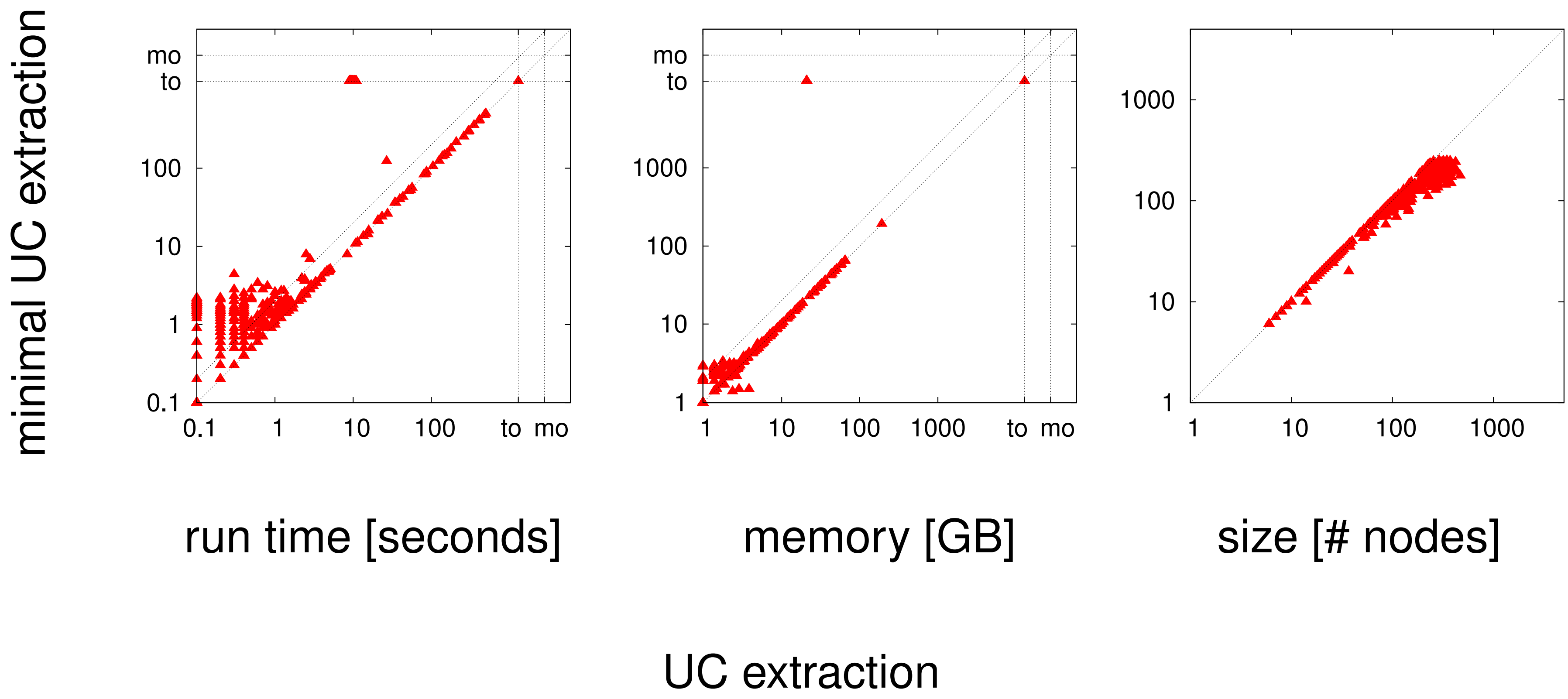
Family	Description	a	b	c	d	Source
			Category **application**			
alaska_lift	Elevator specifications	75 /	72 /	72	4605	[Har05, DW+08]
anzu_genbuf	Generalized buffer	16 /	16 /	16	1924	[Blo+07]
forobots	Model of a robot with properties	25 /	25 /	25	635	[BDF09]
			Category **crafted**			
schup._O1form.	Exponential behavior in some solvers	27 /	27 /	27	4006	[SD11]
schup._O2form.	Exponential behavior in some solvers	8 /	8 /	8	91	[SD11]
schuppan_phltl	Temporal variant of pigeonhole	4 /	4 /	4	125	[SD11]
			Category **random**			
rozier_formulas	Obtained by generating a syntax tree	62 /	62 /	62	157	[RV10]
trp	Obtained by lifting propositional CNF into fixed temporal structure	397 /	397 /	330	1421	[HS02]

a: # solved without UC extraction
b: # solved with extraction of UCs
c: # solved with extraction of minimal UCs
d: |largest solved without UC extraction|

Overhead of UC Extraction

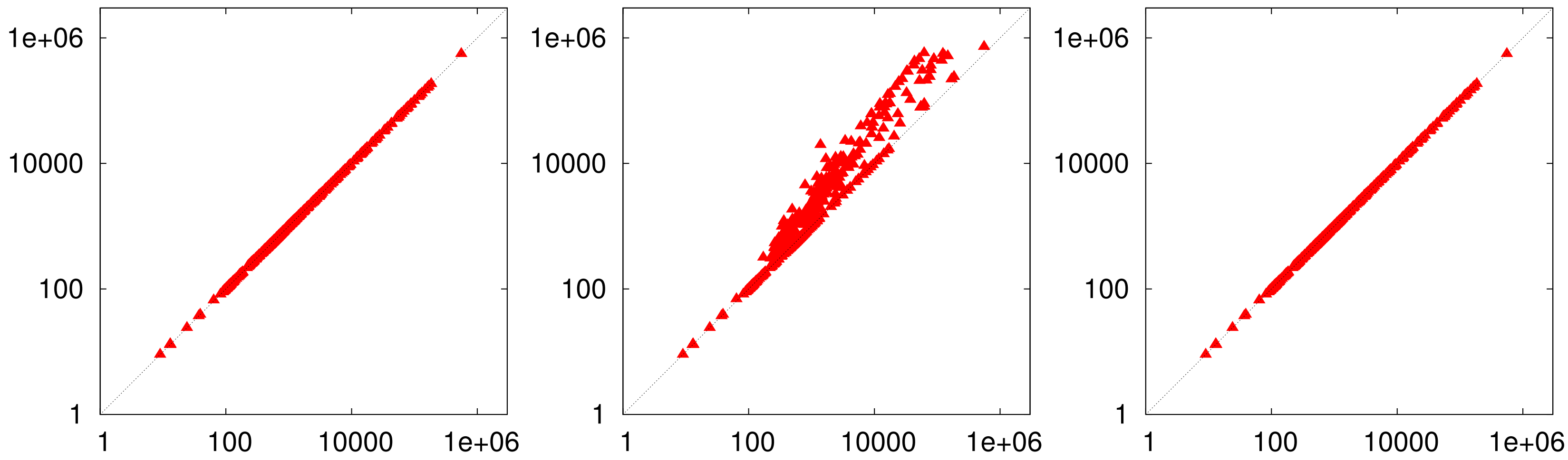


Overhead of Minimal UC Extraction



Author: V. Schuppan

Benefit of Optimizations 1



Shown: peak size of resolution graph [# vertices + # edges]

X-axes: all optimizations enabled

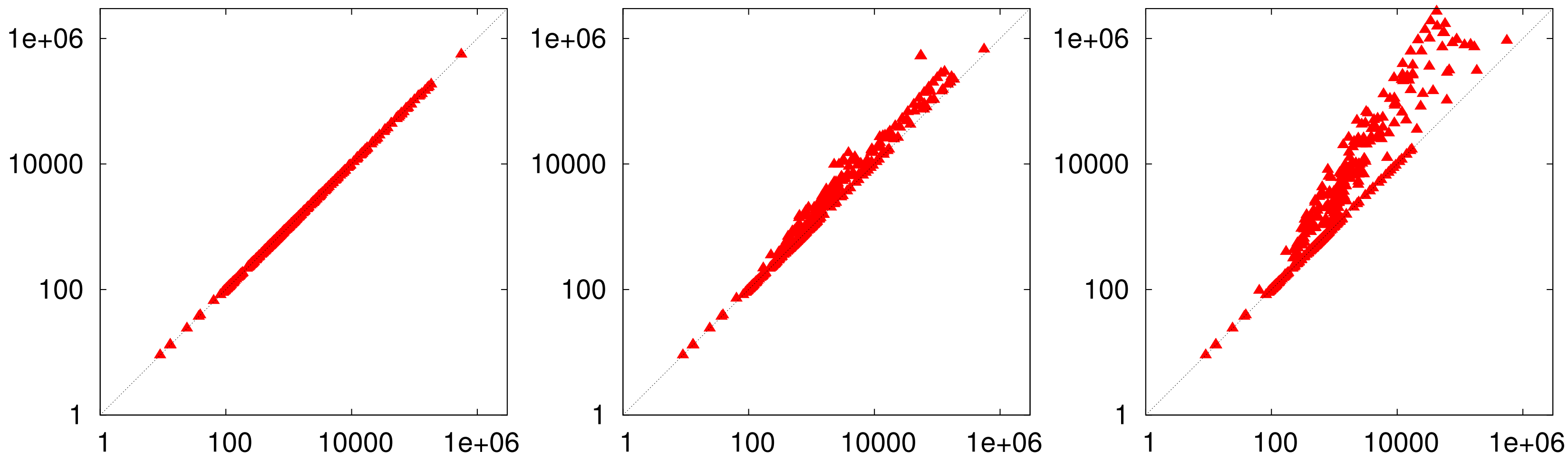
Y-axes:

left include premise of aug2

center include premise 1 of BFS-loop-it-init-c

right include premise 2 of BFS-loop-it-init-c

Benefit of Optimizations 2



Shown: peak size of resolution graph [# vertices + # edges]

X-axes: all optimizations enabled

Y-axes:

left include premise 2 of BFS-loop-conclusion2

center disable pruning of resolution graph between loop searches

right disable all optimizations

Author: V. Schuppan

Outlook: UCs with Sets of Time Points 1

Intuition: replace occurrences of subformulas at specific time points with 1 or 0 depending on polarity (rather than always as before).

Simple example:

$$(\underset{\{1\}}{\mathbf{G}}\ p) \underset{\{0\},\{0\}}{\wedge} (\underset{\{1\}}{\mathbf{X}} \underset{\{1\}}{\neg} p)$$

The p operand of the $\mathbf{G}$ operator "matters" only at time point 1. Other subformulas also "matter" only at time points 0 or 1.

Complex example:

$$p \underset{\{0\},\{0\}}{\wedge} ((\underset{2\cdot\mathbb{N}}{\mathbf{G}} (p \underset{2\cdot\mathbb{N},2\cdot\mathbb{N}}{\rightarrow} \underset{2\cdot\mathbb{N}+1}{\mathbf{X}} \underset{2\cdot\mathbb{N}+2}{\mathbf{X}} p)) \underset{\{0\},\{0\}}{\wedge}$$

$$(\underset{\mathbb{N}}{\mathbf{F}} ((\underset{2\cdot\mathbb{N}}{\neg} p) \underset{2\cdot\mathbb{N},2\cdot\mathbb{N}+1}{\wedge} \underset{2\cdot\mathbb{N}+2}{\mathbf{X}} \underset{2\cdot\mathbb{N}+2}{\neg} p)))$$

$$\left.\begin{array}{l}\text{1st and 2nd conjunct: } p \text{ must be 1 at even time points}\\ \text{3rd conj.: } p \text{ must eventually be 0 two time points in a row}\end{array}\right\} \text{unsat!}$$

Outlook: UCs with Sets of Time Points 2

Some inference rules shift some premises 1 time step into the future.

For example, when using $\mathbf{G}(p \vee q)$ and $\mathbf{XG}(\neg p \vee r)$ to derive $\mathbf{XG}(q \vee r)$, the first premise is shifted.

Fix the empty clause to happen at time point 0. For each input clause c, for each path on which c is backward reachable from $\Box$, count the number of time steps.

Note: loops in the resolution graph complicate the computation.

The End

Summary

Suggested, implemented, and evaluated a method to extract UCs for LTL from a single run of a solver.

UC extraction can be performed efficiently.

Resulting UCs are signficantly smaller than input formulas.

Optimizations help to keep resolution graph small.

Future Work

Use solvers based on SAT or BDDs.

Investigate other temporal logics.

Extend to unrealizable cores.

Thanks to

... you for your attention,

... B. Konev and M. Ludwig for making `TRP++` and `TSPASS` available,

... A. Cimatti for bringing up the subject of temporal resolution.

Questions?

http://www.schuppan.de/viktor/time13/

References

Awa+12 A. Awad, R. Goré, Z. Hou, J. Thomson, and M. Weidlich. An Iterative Approach to Synthesize Business Process Templates from Compliance Rules. Inf. Syst. 37.8, 2012.

Bak+93 R. Bakker, F. Dikker, F. Tempelman, and P. Wognum. Diagnosing and Solving Over-Determined Constraint Satisfaction Problems. IJCAI'93.

BDF09 A. Behdenna, C. Dixon, and M. Fisher. Deductive Verification of Simple Foraging Robotic Behaviours. International Journal of Intelligent Computing and Cybernetics, 2009.

Bee+01 I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient Detection of Vacuity in Temporal Model Checking. Formal Methods in System Design 18.2, 2001.

bgl http://www.boost.org/doc/libs/release/libs/graph/.

Blo+07 R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, Compile, Run: Hardware from PSL. COCV'07.

BS01 R. Bruni and A. Sassano. Restoring Satisfiability or Maintaining Unsatisfiability by finding small Unsatisfiable Subformulae. SAT'01.

CD91 J. Chinneck and E. Dravnieks. Locating Minimal Infeasible Constraint Sets in Linear Programs. ORSA Journal on Computing 3.2, 1991.

Cim+02 A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. CAV'02.

Cim+08 A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic Information for Realizability. VMCAI'08.

DW+08 M. De Wulf, L. Doyen, N. Maquet, and J.-F. Raskin. Antichains: Alternative Algorithms for LTL Satisfibility and Model-Checking. TACAS'08.

EF06 C. Eisner and D. Fisman. A Practical Introduction to PSL. Springer, 2006.

FDP01 M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. ACM Trans. Comput. Log. 2.1, 2001.

Fis91 M. Fisher. A Resolution Method for Temporal Logic. IJCAI'91.

Heu+95 A. Heuerding, G. Jëger, S. Schwendimann, and M. Seyfried. Propositional Logics on the Computer. TABLEAUX'95.

HK03 U. Hustadt and B. Konev. TRP++ 2.0: A Temporal Resolution Prover. CADE'03.

HK04 U. Hustadt and B. Konev. TRP++: A Temporal Resolution Prover. Collegium Logicum, Vol. 8, 2004.

Har05 A. Harding. Symbolic Strategy Synthesis For Games With LTL Winning Conditions. PhD thesis. University of Birmingham, 2005.

HS02 U. Hustadt and R. A. Schmidt. Scientific Benchmarking with Temporal Logic Decision Procedures. KR'02.

Jos12 N. Josuttis. The C++ Standard Library: A Tutorial and Reference. Second Edition. Addison Wesley, 2012.

MS10 J. Marques Silva. Minimal Unsatisfiability: Models, Al-gorithms and Applications (Invited Paper). ISMVL'10.

PA06 M. Pesic and W. van der Aalst. A Declarative Approach for Flexible Business Processes Management. Business Process Management Workshops. 2006.

PG86 D. Plaisted and S. Greenbaum. A Structure-Preserving Clause Form Translation. J. Symb. Comput. 2.3, 1986.

Pil+06 I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal Analysis of Hardware Requirements. DAC'06.

pltl http://users.cecs.anu.edu.au/rpg/PLTLProvers/

run A. Biere and T. Jussila. Benchmark Tool Run. URL: http://fmv.jku.at/run/

RV10 K. Rozier and M. Vardi. LTL Satisfiability Checking. STTT, 12(2), 2010.

Sch12 V. Schuppan. Towards a Notion of Unsatisfiable and Unrealizable Cores for LTL. Sci. Comput. Program. 77.7-8, 2012.

SD11 V. Schuppan and L. Darmawan. Evaluating LTL Satisfiability Solvers. ATVA'11.

Shl+03 I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri. Debugging Overconstrained Declarative Models Using Unsatisfiable Cores. ASE'03.

SL95 A. Stepanov and M. Lee. The Standard Template Library. Tech. rep. 95-11 (R.1), HP Laboratories, Nov. 1995.

SLL02 J. Siek, L. Lee, and A. Lumsdaine. The Boost Graph Library - User Guide and Reference Manual. Pearson/Prentice Hall, 2002.

trp++ http://www.csc.liv.ac.uk/ konev/software/trp++/.

Wha94 D. Whalley. Automatic Isolation of Compiler Errors. ACM Trans. Program. Lang. Syst. 16.5, 1994.

VG02 A. Van Gelder. Extracting (Easily) Checkable Proofs from a Satisfiability Solver that Employs both Preorder and Postorder Resolution. AMAI'02.

ZH02 A. Zeller and R. Hildebrandt. Simplifying and Isolating Failure-Inducing Input. IEEE Trans. Software Eng. 28.2, 2002.