SAT & QBF in Formal Verification

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

RISC Seminar

Schloß Hagenberg

March 14, 2005

Overview

1. SAT
   - DPLL
   - Decision Heuristics and Learning
2. Bounded Model Checking
3. QBF
   - QBF for Symbolic Traversal
   - State-of-the-Art in QBF Solvers
   - Resolve & Expand

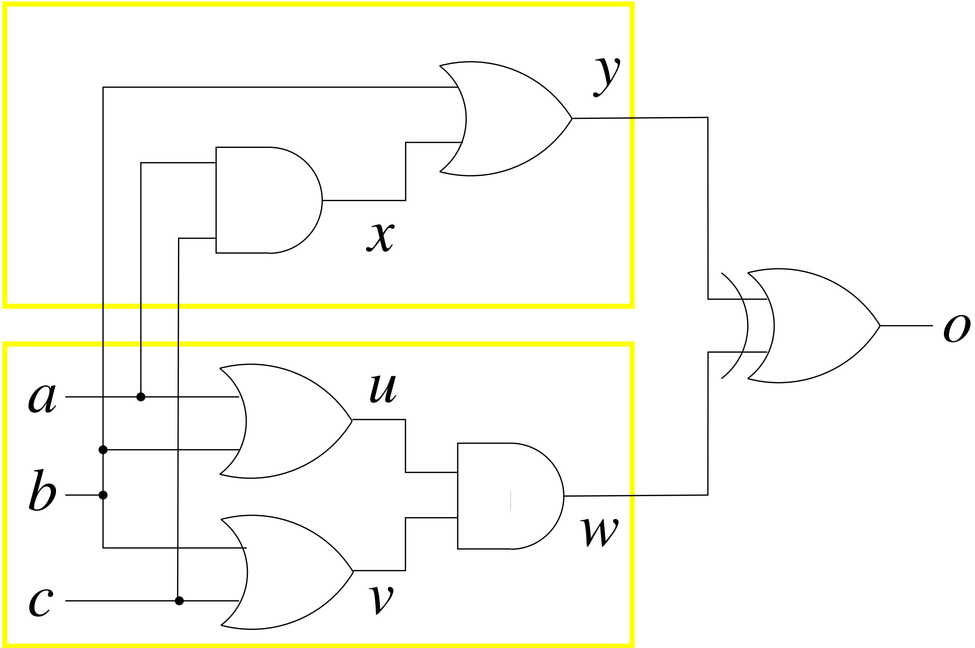
- input formula in conjunctive normal form (CNF)
  - a formula in **CNF** is a conjunction of clauses
  - each **clause** a disjunction of literals
  - a **literal** is positive (v) or negated boolean variable ($\neg v$)

$$(\neg r \vee v) \wedge (s \vee v) \wedge (x \vee y \vee v) \wedge (\neg v \vee r) \wedge (\neg v \vee \neg x \vee \neg y \vee \neg r)$$

- SAT = check whether formula in CNF is satisfiable
  (satisfiable = exists assignments which makes the formula true)
  - **the** NP complete problem
  - can be restricted (also in practice) to clauses of length 3
  - equivalent to check formula or circuit satisfiability

Tseitin Transformation: Circuit to CNF

equivalence checking problem



constraints

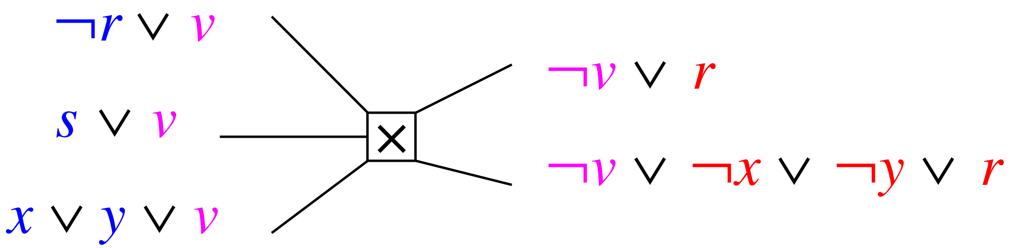
$$
\begin{array}{l}
o \;\wedge \\
(x \leftrightarrow a \wedge c) \;\wedge \\
(y \leftrightarrow b \vee x) \;\wedge \\
(u \leftrightarrow a \vee b) \;\wedge \\
(v \leftrightarrow b \vee c) \;\wedge \\
(w \leftrightarrow u \wedge v) \;\wedge \\
(o \leftrightarrow y \oplus w)
\end{array}
$$

$o \wedge (x \rightarrow a) \wedge (x \rightarrow c) \wedge (x \leftarrow a \wedge c) \wedge \; \ldots$ implications

$o \wedge (\overline{x} \vee a) \wedge (\overline{x} \vee c) \wedge (x \vee \overline{a} \vee \overline{c}) \wedge \; \ldots$ clauses

Variable Elimination by Resolution

original clauses in which v or $\neg v$ occurs:



add non-trivial resolvents:

$(s \vee r)$, $(x \vee y \vee r)$, and $(s \vee \neg x \vee \neg y \vee r)$

remove original clauses

Pure Literals

- pure literal l in a CNF f
  - l occurs in f
  - $\neg l$ does not occur in f
- clauses with pure literals can be removed
  - result $f\{l/1\}$
  - $f\{l/0\} \Rightarrow f\{l/1\}$
  - stronger semantic criteria possible (e.g. autarkies)
- pure literal reduction as satisfiability preserving transformation

DP for SAT

[DavisPutnam60]

```
dp-sat()
  forever
    boolean-constraint-propagation()
    if contains-empty-clause() then return unsatisfiable
    remove-clauses-with-pure-literals()
    if no-clause-left() then return satisfiable
    v := next-not-eliminated-variable()
    C_v := clauses-containing(v)
    C_¬v := clauses-containing(¬v)
    C' := ∅
    forall c_v ∈ C_v do
      forall c_¬v ∈ C_¬v do
        c' := resolve(v, c_v, c_¬v)
        if non-trivial(c') then C' := C' ∪ {c'}
    replace C_v ∪ C_¬v by C'
```

DPLL for SAT

[DavisLogemannLoveland62]

Trade Space for Time

dpll-sat(*Assignment* S)
  boolean-constraint-propagation()
  if contains-empty-clause() **then return** *unsatisfiable*
  if no-clause-left() **then return** *satisfiable*
  v := next-unassigned-variable()
  return dpll-sat(S $\cup \{v \mapsto$ *false*$\}$) $\vee$ dpll-sat(S $\cup \{v \mapsto$ *true*$\}$)

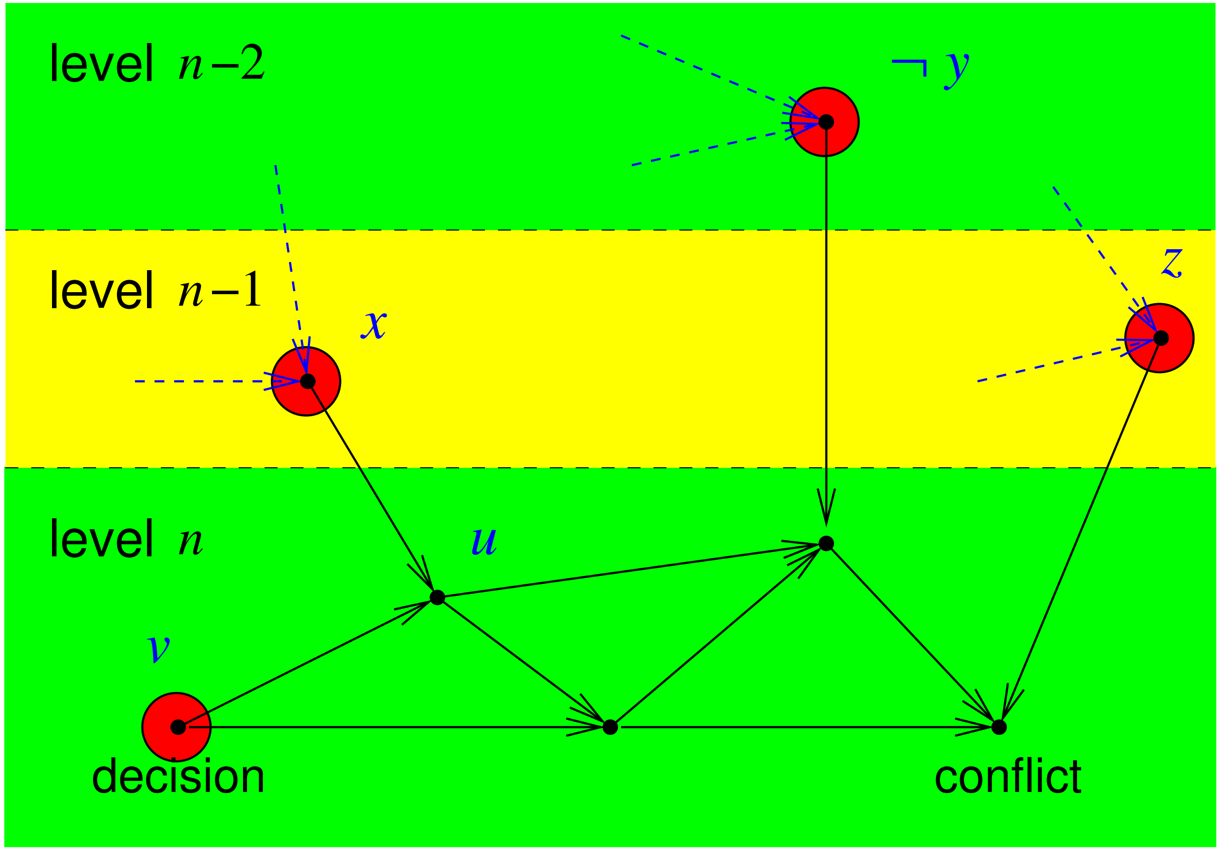
(pure literal rule omitted)

Historical Perspective I

- early 90ies
  - focus on decision heuristics
  - 1st order heuristics
    - derived from current assignment plus formula
    - example: dynamic independent literal sum (DLIS)
    - does not take search history into account ($\Rightarrow$ 1st order)
- mid 90ies
  - non-chronlogical backtracking, learning, conflict driven assignment

  Solvers: RELSAT, GRASP, SATO

Implication Graph and Learning



learned clause: $(\neg v \lor \neg x \lor y \lor \neg z)$

Historical Perspective II

- end of 90ies
  - SAT solvers became mature enough to be used in various applications
  - e.g. in formal verification: bounded model checking (BMC)
- since 2000
  - wide spread industrial usage of SAT solvers in circuit verification
  - improved lazy data structures, 2nd order decision heuristics

    Solvers: ZCHAFF, BERKMIN
  - regular SAT solver competition

2nd Order Decision Heuristics

- take search history into account
  - focus on literals that recently contributed to conflicts
  - pioneered by CHAFF's Variable State Independent Decaying Sum (VSIDS):
    1. increase score of literals in learned clauses
    2. exponentially decrease all scores over time
    3. pick unassigned variable with largest score
- works incredibly well in practice, but it is (still) unclear why

Explicit/Symbolic Checking

- **model checking** is about verifying temporal properties of systems algorithmically
  - builds on Pnueli's idea on using temporal logic for specification purposes
  - **explicit** model checking represents states explicitly [EmersonClarke81]
- **state explosion problem**, particulary in hardware verification:
  - state space grows exponentially with the size of the system description
  - symmetry or partial order reduction as one solution
- **symbolic model checking**
  - symbolic representations for sets of states to combat the state explosion problem
  - originally with binary decision diagrams (BDDs)
    [CoudertMadre89,BurchClarkeMcMillanDillHwang90,McMillan93]

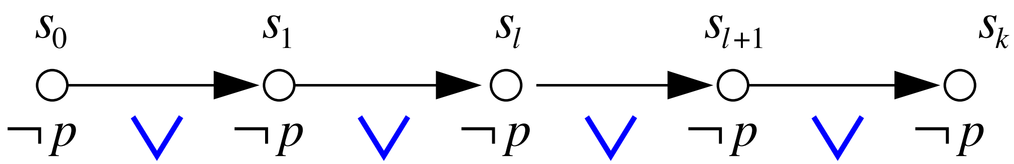
Bounded Model Checking

[BiereClarkeCimattiZhu99]

- **motivation**: leverage improvements of SAT technology for model checking
  - BDD based model checking did and does not scale as much as necessary
  - SAT seems to be more *robust* than BDDs
- original **idea**: shift focus towards falsification instead of verification
  - search for counter example traces of a certain length k
  - reformulate existence of a counter example of length k as SAT problem
- impact:
  - industry uses simulation, then bounded and finally BDD based model checking
  - accelerated interest in SAT technology

Bounded Model Checking Safety

checking safety property $\mathbf{G}p$ for a bound k as SAT problem:

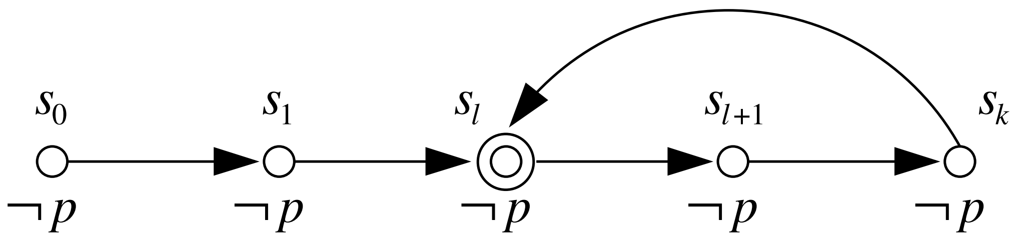


$$I(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

check occurrence of $\neg p$ in the first k states

Bounded Model Checking Liveness

generic counter example trace of length k for liveness $\mathbf{F}p$



$$I(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_k, s_{k+1}) \wedge \bigvee_{l=0}^{k} s_l = s_{k+1} \wedge \bigwedge_{i=0}^{k} \neg p(s_i)$$

(however we recently showed that liveness can always be reformulated as safety [BiereArthoSchuppan02])

Completeness in Bounded Model Checking

- find bounds on the maximal length of counter examples
  - also called **completeness threshold**
  - exact bounds are hard to find $\Rightarrow$ approximations
- induction
  - use of inductive invariants (manually generated)
  - generalization of inductive invariants: **pseudo induction** or k-induction
- use SAT for quantifier elimination as with BDDs
  - then model checking becomes fixpoint calculation
  - alternatively use approximate elimination (as in McMillan's interpolation)
- or in an abstraction/refinement loop

T boolean formula encoding of a (finite transition) relation

$$[[T]] \subseteq \{0,1\}^n \times \{0,1\}^n$$

Transitive Closure

$$T^* \equiv T^{2^n}$$

Standard Linear Unfolding

$$T^{i+1}(s,t) \equiv \exists\, m.\, T^i(s,m) \wedge T(m,t)$$

Iterative Squaring via Copying

$$T^{2\cdot i}(s,t) \equiv \exists\, m.\, T^i(s,m) \wedge T^i(m,t)$$

Non Copying Iterative Squaring

$$T^{2\cdot i}(s,t) \equiv \exists\, m.\, \forall c.\, \exists\, l,r.\ (c \rightarrow (l,r) = (s,m)) \wedge (\overline{c} \rightarrow (l,r) = (m,t)) \wedge T^i(l,r)$$

DPLL for SAT and QBF

```
dpll-sat(Assignment S)                                   [DavisLogemannLoveland62]
    boolean-constraint-propagation()
    if contains-empty-clause() then return false
    if no-clause-left() then return true
    v := next-unassigned-variable()
    return  dpll-sat(S ∪ {v ↦ false}) ∨ dpll-sat(S ∪ {v ↦ true})
```

```
dpll-qbf(Assignment S)                                   [CadoliGiovanardiSchaerf98]
    boolean-constraint-propagation()
    if contains-empty-clause() then return false
    if no-clause-left() then return true
    v := next- outermost -unassigned-variable()
    @ := is-existential(v) ? ∨ : ∧
    return  dpll-sat(S ∪ {v ↦ false})  @  dpll-sat(S ∪ {v ↦ true})
```

The Crux of QBF

Why is QBF harder than SAT?

$$\models \quad \forall x \, . \, \exists y \, . \, (x \leftrightarrow y)$$

$$\not\models \quad \exists y \, . \, \forall x \, . \, (x \leftrightarrow y)$$

Decision Order Matters!

State-of-the-Art

- almost all implementations are QBF-enhanced DPLL: [Cadoli...98] [Rintanen01]
  - recently **learning** was added [Giunchiglia...01] [Letz01] [ZhangMalik02]
  - all deterministic solvers (except one) in QBF-Evaluation'03 were DPLL based
  - top-down: split on variables from the outside to the inside
- multiple quantifier elimination procedures:
  - **enumeration** [PlaistedBiereZhu03] [McMillan02]
  - **expansion** [Aziz-Abdulla...00] [WilliamsBiere...00] [AyariBasin02]
  - bottom-up: eliminate variables from the inside to the outside
- **q-resolution** [Kleine-Büning...95]

Forall Reduction and Q-Resolution

- **collect** variables in scopes, **order** variables and scopes according to nesting depth:

$$\underbrace{\exists\, a,b,c,d.}_{\text{scope 0}} \quad \underbrace{\forall x,y,z.}_{\text{scope 1}} \quad \underbrace{\exists\, r,s,t.}_{\text{scope 2}} \quad (c \vee d)(a \vee \overline{c} \vee \overline{x} \vee y)(\overline{a} \vee x \vee s)(t \vee \ldots)\ \cdots$$

attach clauses to the scope of its innermost variables

- **remove** innermost universal literals in clauses attached to universal scopes:

$$(a \vee \overline{c} \vee \overline{x} \vee y) \quad \text{simplifies to} \quad (a \vee \overline{c})$$

- q-resolution = resolution + forall reduction

- all clauses are forall reduced

  $\Rightarrow$ innermost scope is always existential

  $\Rightarrow$ no clauses attached to universal scopes

- normalized structure of quantified CNF:

$$\Omega(S_1)\, S_1 \,.\quad \Omega(S_2)\, S_2 \,.\quad \ldots \quad \forall S_{m-1} \,.\quad \exists\, S_m \,.\quad f \wedge g \qquad m \geq 2$$

$f \equiv$ clauses of scope S_m

$g \equiv$ clauses of outer scopes $S_i, \quad i < m-1$

$S_\exists \equiv S_m$

$S_\forall \equiv S_{m-1}$

Algorithm

<u>resolve-and-expand</u>()

forever

simplify()

if contains-empty-clause() **then return** *false*

if no-clause-left() **then return** *true*

if is-propositional() **then return** sat-solve($\emptyset$)

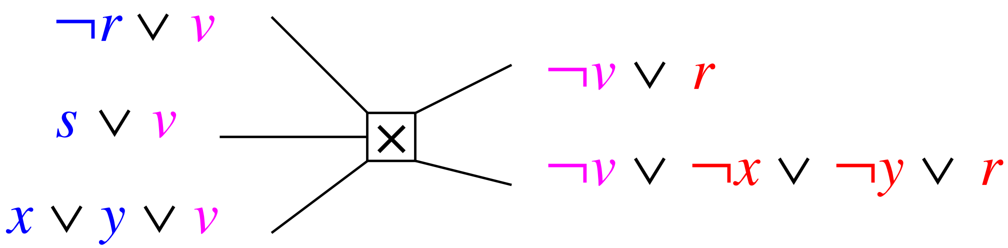
v := schedule-cheapest-to-eliminate-variable()

if is-existential(v) **then** resolve(v)

if is-universal(v) **then** expand(v)

Resolve

original clauses in which v or $\neg v$ occurs:



add forall reduced non-trivial resolvents:

$(s \vee r)$, $(x \vee y \vee r)$, and $(s \vee \neg x \vee \neg y \vee r)$

remove original clauses

Expand

one-to-one mapping of variables: $u \in S_\exists$ mapped to $u' \in S'_\exists$

before expansion:

$$\Omega(S_1)\, S_1 \,.\quad \Omega(S_2)\, S_2 \,.\quad \dots\quad \forall S_\forall \qquad .\quad \exists\, S_\exists \qquad . \qquad f \wedge g$$

after expansion:

$$\Omega(S_1)\, S_1 \,.\quad \Omega(S_2)\, S_2 \,.\quad \dots\quad \forall(S_\forall - \{v\}).\quad \exists(S_\exists \cup S'_\exists).\quad f\{v/0\} \wedge f'\{v/1\} \wedge g$$

Schedule

- elimination cost: number of expected added literals

$$
\begin{aligned}
o(l) &\equiv \text{number of clauses with literal } l \\
s(l) &\equiv \text{sum of lengths of clauses with literal } l \\
s(S) &\equiv \text{sum lengths of clauses with scope } S
\end{aligned}
$$

- expansion cost: $\mathrm{s}(\mathrm{S}_\exists) \quad - \quad \Big(\mathrm{s}(\mathrm{v}) + \mathrm{s}(\neg\mathrm{v}) + \mathrm{o}(\mathrm{v}) + \mathrm{o}(\neg\mathrm{v})\Big)$

- resolution cost: $\mathrm{o}(\neg\mathrm{v}) \cdot \Big(\mathrm{s}(\mathrm{v}) - \mathrm{o}(\mathrm{v})\Big) + \mathrm{o}(\mathrm{v}) \cdot \Big(\mathrm{s}(\neg\mathrm{v}) - \mathrm{o}(\neg\mathrm{v})\Big) - \Big(\mathrm{s}(\mathrm{v}) + \mathrm{s}(\neg\mathrm{v})\Big)$

	benchmark family	#inst	**decide**	**qube**	**semprop**	*expand*	**quantor**
1	adder*	16	2	2	2	1	**<u>3</u>**
2	Adder2*	14	2	2	2	2	**<u>3</u>**
3	C[0-9]*	27	2	3	2	3	**<u>4</u>**
4	CHAIN*	11	10	7	**11**	4	**11**
5	comp*	5	4	4	**5**	**5**	**5**
6	flip*	7	6	**7**	**7**	**7**	**7**
7	impl*	16	12	**16**	**16**	**16**	**16**
8	k*	171	77	91	97	60	**<u>108</u>**
9	mutex*	2	1	**2**	**2**	**2**	**2**
10	robots*	48	0	**36**	**36**	15	24
11	term1*	4	2	**3**	**3**	1	**3**
12	toilet*	260	187	**260**	**260**	259	259
13	TOILET*	8	**8**	6	**8**	**8**	**8**
14	tree*	12	10	**12**	**12**	8	**12**
#(among best in family)			**1**	**7**	**10**	**5**	**12**
#(single best in family)			**<u>0</u>**	**<u>0</u>**	**<u>0</u>**	**<u>0</u>**	**<u>4</u>**

(families with no difference and two actually random families removed)

Simplify

- resolve quadratic in number of occurrences, expand may double the size

  $\Rightarrow$ simplify CNF as much as possible before elimination

- standard simplification: **unit propagation**, **pure literal rule**, **forall reduction**

- **equivalence reasoning**: extract bi-implications and substitute variables

$$\forall x\,.\exists y\,.(x \vee y)(x \rightarrow y)(y \rightarrow x) \quad \equiv \quad \forall x\,.\exists y\,.(x \vee y)(x = y) \quad \equiv \quad \forall x\,.\exists y\,.(x \vee x) \quad \equiv \quad 0$$

- **subsumption**: remove subsumed clauses
  - backward subsumption is checked on-the-fly whenever a clause is added
  - forward subsumption is expensive and only checked before expensive operations

Solved Hard Instances of SAT'03 QBF Evaluation: QUANTOR

	hard instance	time	space	∀	∃	units	pure	subsu.	subst.	∀red.
1	Adder2-6-s	29.6	19.7	90	13732	126	13282	174081	0	37268
2	adder-4-sat	0.2	2.8	42	1618	0	884	6487	0	960
3	adder-6-sat	36.6	22.7	90	13926	0	7290	197091	0	54174
4	C49*1.*_0_0*	27.9	13.3	1	579	0	0	48	84	0
5	C5*1.*_0_0*	56.2	16.0	2	2288	10	0	4552	2494	0
6	k_path_n-15	0.1	0.8	32	977	66	82	2369	2	547
7	k_path_n-16	0.1	0.8	34	1042	69	85	2567	2	597
8	k_path_n-17	0.1	0.9	36	1087	72	100	3020	2	639
9	k_path_n-18	0.1	0.9	36	1146	76	106	3242	2	725
10	k_path_n-20	0.1	0.9	38	1240	84	149	3967	2	855
11	k_path_n-21	0.1	1.0	40	1318	84	130	4470	2	909
12	k_t4p_n-7	15.5	105.8	43	88145	138	58674	760844	8	215
13	k_t4p_p-8	5.8	178.6	29	12798	206	5012	85911	4	138
14	k_t4p_p-9	0.3	4.5	32	4179	137	1389	23344	10	142
15	k_t4p_p-10	27.9	152.9	35	130136	193	63876	938973	4	137
16	k_t4p_p-11	86.0	471.5	38	196785	204	79547	1499430	4	140
17	k_t4p_p-15	84.6	354.7	50	240892	169	181676	1336774	9	226
18	k_t4p_p-20	3.6	16.1	65	27388	182	21306	197273	11	325

time in seconds, space in MB

	hard instance	time	space	∀
1	Adder2-6-s	(12.2)	m.o.	–
2	adder-4-sat	(12.1)	m.o.	–
3	adder-6-sat	(13.0)	m.o.	–
4	C49*1.*_0_0*	98.3	40.8	1
5	C5*1.*_0_0*	357.0	45.6	2
6	k_path_n-15	(16.5)	m.o.	–
7	k_path_n-16	(16.6)	m.o.	–
8	k_path_n-17	(16.2)	m.o.	–
9	k_path_n-18	(16.8)	m.o.	–
10	k_path_n-20	(21.4)	m.o.	–
11	k_path_n-21	(21.0)	m.o.	–
12	k_t4p_n-7	(16.8)	m.o.	–
13	k_t4p_p-8	(21.4)	m.o.	–
14	k_t4p_p-9	(21.2)	m.o.	–
15	k_t4p_p-10	(17.3)	m.o.	–
16	k_t4p_p-11	(17.3)	m.o.	–
17	k_t4p_p-15	(21.3)	m.o.	–
18	k_t4p_p-20	(20.9)	m.o.	–

time in seconds, space in MB, m.o. = memory out ($>$ 1 GB)