A BDD/SAT Solver for Formal Verification Applications

John Franco, Robert Price, John Schlipf, Jeff Ward, Sean Weaver

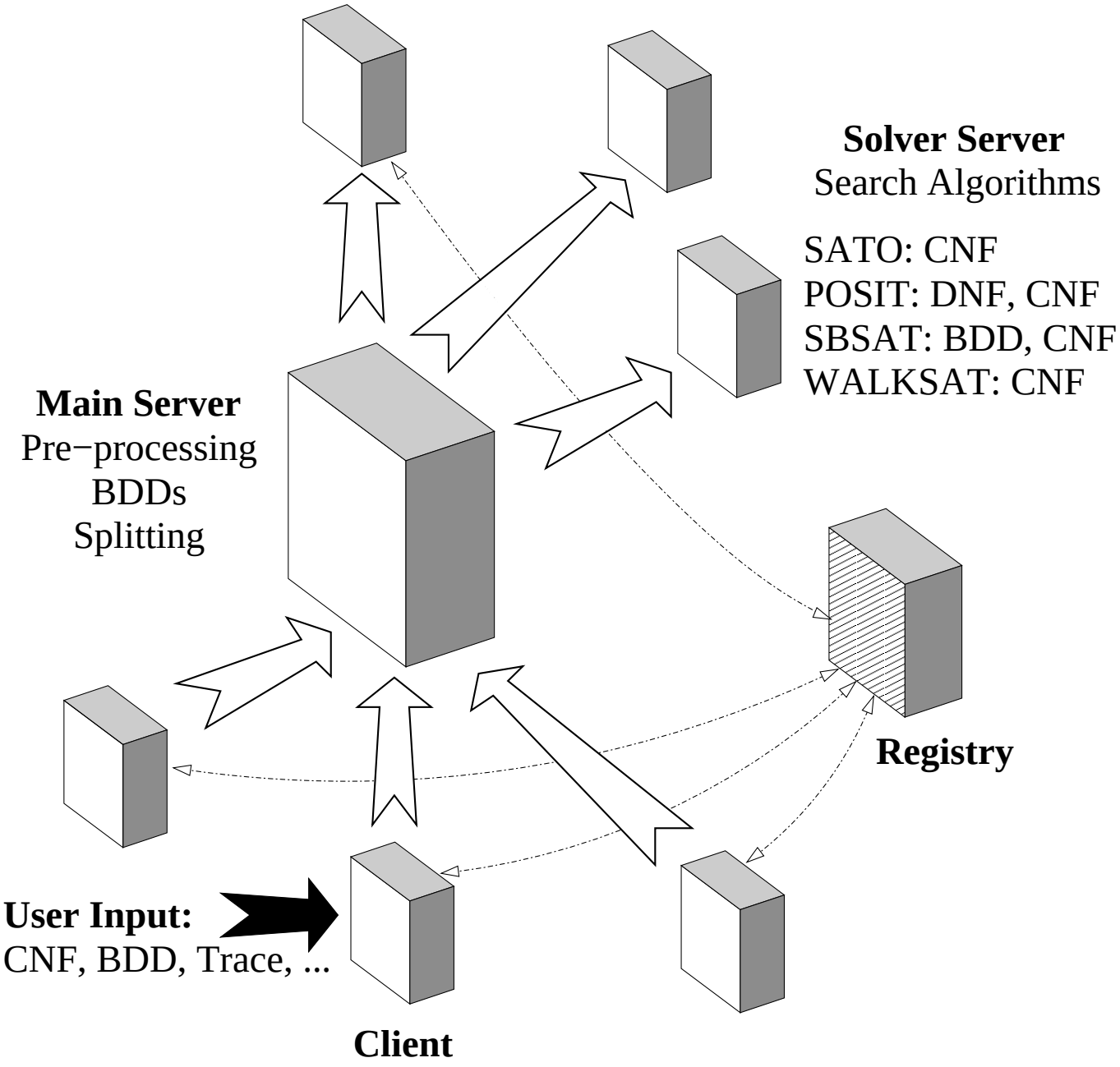
ECECS, University of Cincinnati

Michael Dransfield, Mark Vanfleet

National Security Agency

Research supported by NSA-R2

Overview


Solver Server
Search Algorithms
SATO: CNF
POSIT: DNF, CNF
SBSAT: BDD, CNF
WALKSAT: CNF
Main Server
Pre−processing
BDDs
Splitting
Registry
User Input:
CNF, BDD, Trace, ...
Client

Combine Best of BDD and Search

Trade Space for Speed

- ❄ Precompute everything: table lookup later
- ❄ Make inferences for search later
- ❄ BDD front end - operate before search

➠ Goal: increase backtracks/second

Stay in the User Domain

- ❄ Information clustered for faster inferencing
- ❄ Avoid translation to CNF

➠ Goal: reduce number of backtracks

Search

- ❄ Exploit input structure and precomputation
- ❄ Drive toward solution or refutation?
- ❄ Discover and save inferences
- ❄ Reuse inferences later to prune search

➠ Goal: reduce number of backtracks

Input Language and Modules

Example:

```
MODULE ww
INPUT
  0, 1, 2;
OUTPUT  out;
STRUCTURE
  4 = not(0);
  5 = not(1);
  6 = not(2);
  7 = or(0, 1, 2);
  t = or(0, 1, 6);
  q = or(not(7), or(r, and(1, 2, 5), t), and(2, z = and(2, 1), 45));
  r = or(0, 5, not(2));
  3 = or(4, z, 2);
  3 = or(4, 1, z);
  3 = tester(f, not(1), 2, 1);
  out = or(4, 5, 6);
  out = new_int_leaf(1);
ENDMODULE
```

Module:

```
MODULE tester
INPUT x, y, z;
OUTPUT a, f;
STRUCTURE
  a = and(or(y, z), x);
  f = or(and(x, z), y);
ENDMODULE
```

Compiled:

```
FUNCTION a x y z
^Aae = or(y, z);
a = and(^Aae, x);

FUNCTION f x y z
^Aae = and(x, z);
f = or(^Aae, y);
```

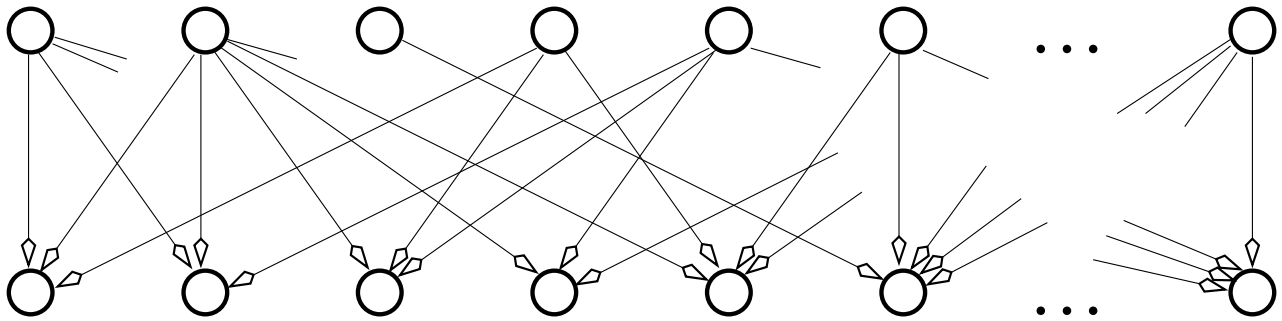
Abridged example input (from CMU webpage)

```
MODULE module-name-changed
INPUT
  ID_EX_RegWrite, ID_EX_MemToReg, _Taken_Branch_1_1, EX_MEM_Jump, ...
OUTPUT  _temp_1252;
STRUCTURE
  _squash_1_1 = or(_Taken_Branch_1_1, EX_MEM_Jump);
  _squash_bar_1_1 = not(_squash_1_1);
  _EX_Jump_1_1 = and(_squash_bar_1_1, ID_EX_Jump);
  _Taken_Branch_9_1 = and(_squash_bar_1_1, ID_EX_Branch, TakeBranchALU_0);
  _squash_9_1 = or(_EX_Jump_1_1, _Taken_Branch_9_1);
  _squash_bar_9_1 = not(_squash_9_1);
  _IF_ID_Flush_bar_1_1 = not(IF_ID_Flush);
  _Reg2Used_1_1 = or(IF_ID_UseData2, IF_ID_Branch, IF_ID_MemWrite, IF_ID_MemToReg);
  _temp_967 = and(_Reg2Used_1_1, e_2_1);
  _temp_968 = or(e_1_1, _temp_967);
  _temp_969 = and(_IF_ID_Flush_bar_1_1, ID_EX_RegWrite, ID_EX_MemToReg, _temp_968);
  _temp_970 = or(_Taken_Branch_1_1, IF_ID_Flush, EX_MEM_Jump, _temp_969);
  _temp_971 = not(_temp_970);
  _temp_972 = and(IF_ID_Jump, _squash_bar_9_1, _temp_971);
  _temp_973 = and(IF_ID_Branch, _squash_bar_9_1, TakeBranchALU_1, _temp_971);
  _temp_974 = or(_temp_972, _temp_973);
  _temp_975 = not(_temp_974);
  _temp_976 = ite(_temp_969, IF_ID_Jump, Jump_0);
  ...
  _temp_1249 = and(_temp_1038, _temp_1066, _temp_1072, _temp_1189, _temp_1246);
  _temp_1250 = or(_temp_1248, _temp_1249);
  _temp_1251 = and(_temp_1233, _temp_1242, _temp_1250);
  _temp_1252 = or(_temp_1161, _temp_1251);
  true_value = new_int_leaf(1);
  are_equal(_temp_1252, true_value); % 1
ENDMODULE
```

Courtesy M.N. Velev, Superscalar Suite 1.0. Available from: http://www.ece.cmu.edu/~ mvelev.

General Input Types

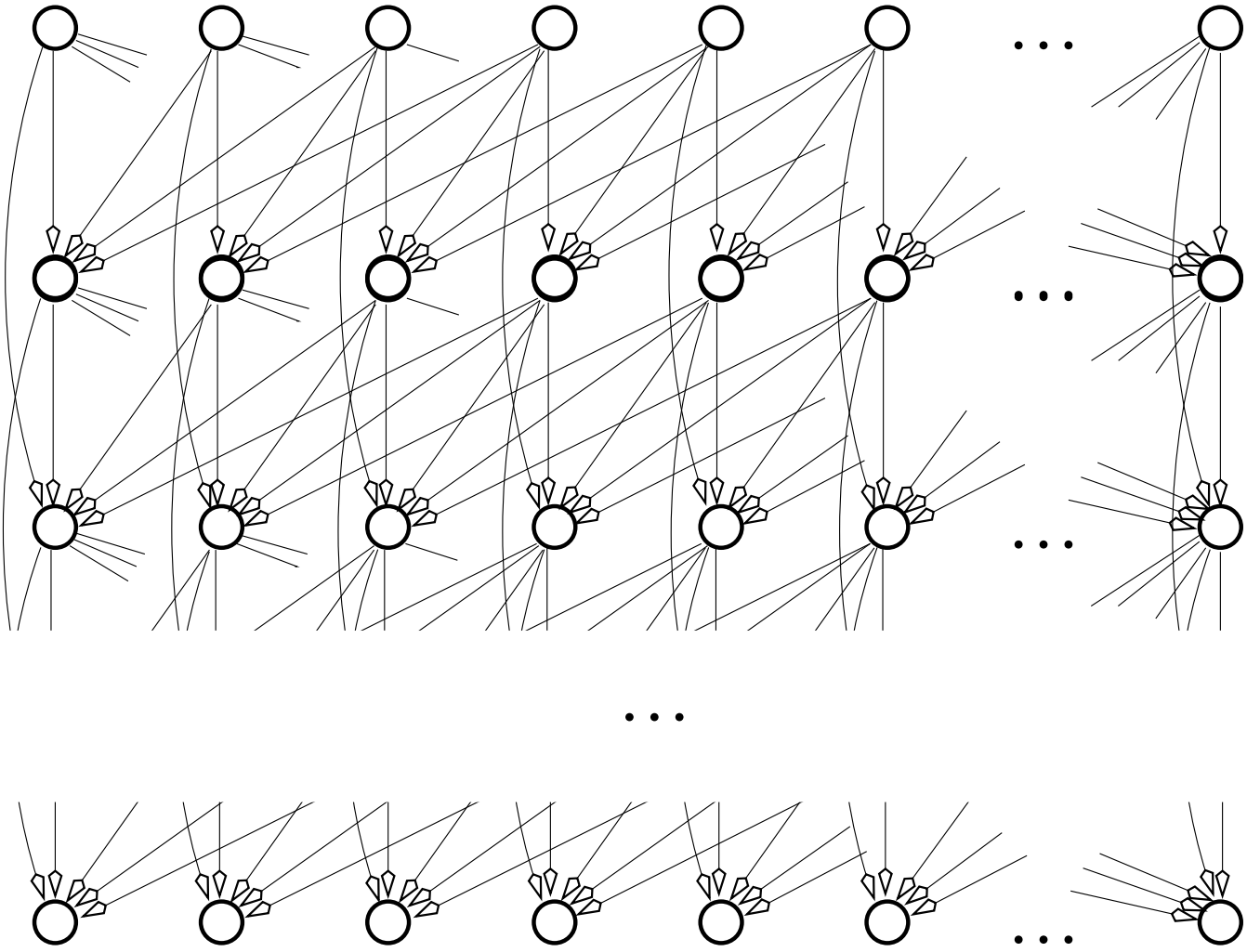
Inputs



Outputs

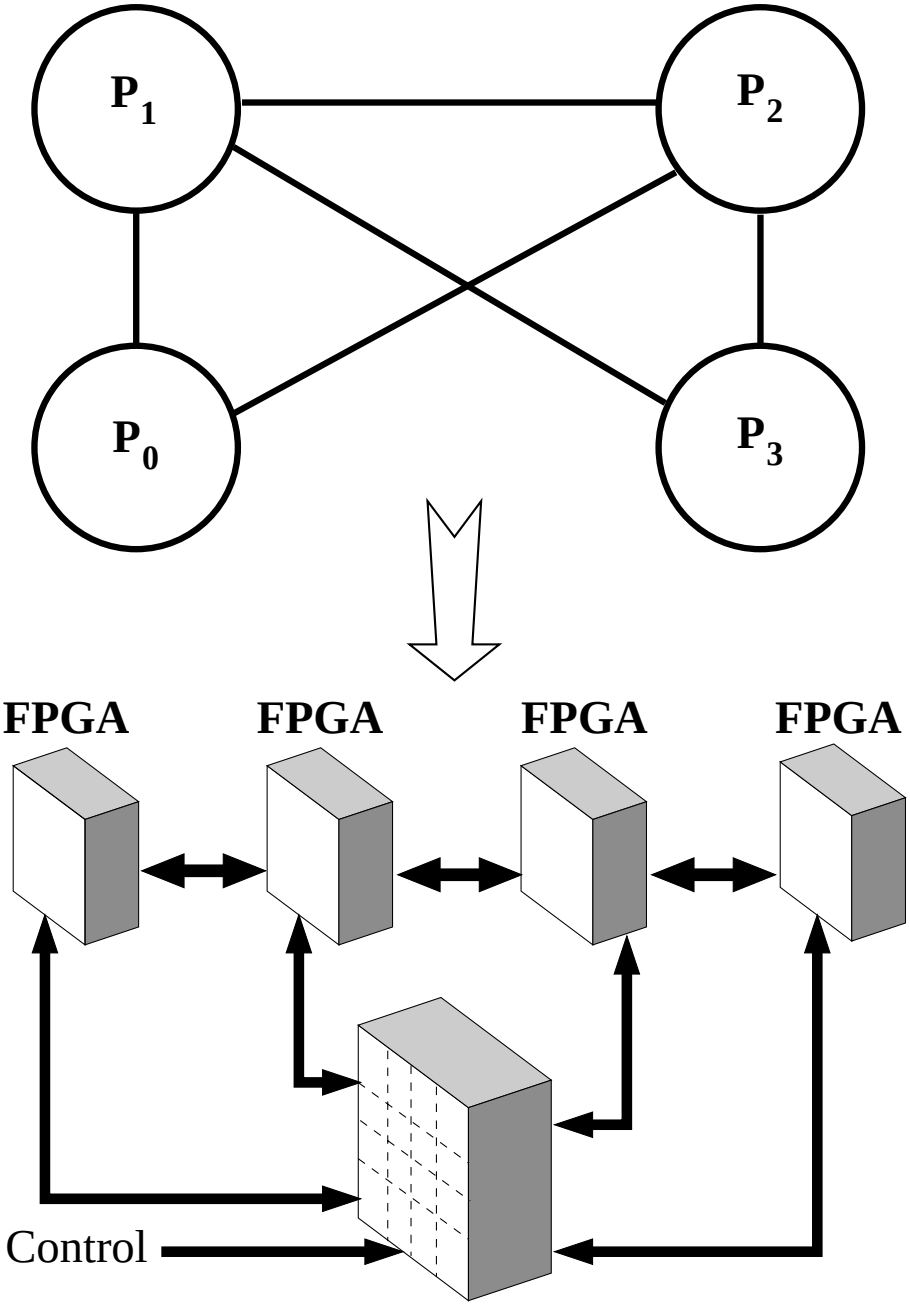
Is there an input consistent with a given output?

Layered Boolean Functions: Input Dependencies



Sliding, width 4 LBF

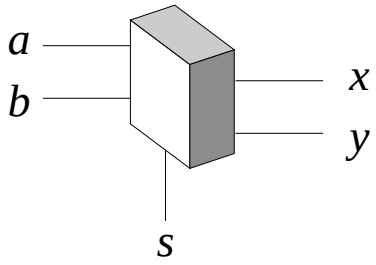
Interconnect Synthesis



Example

Example

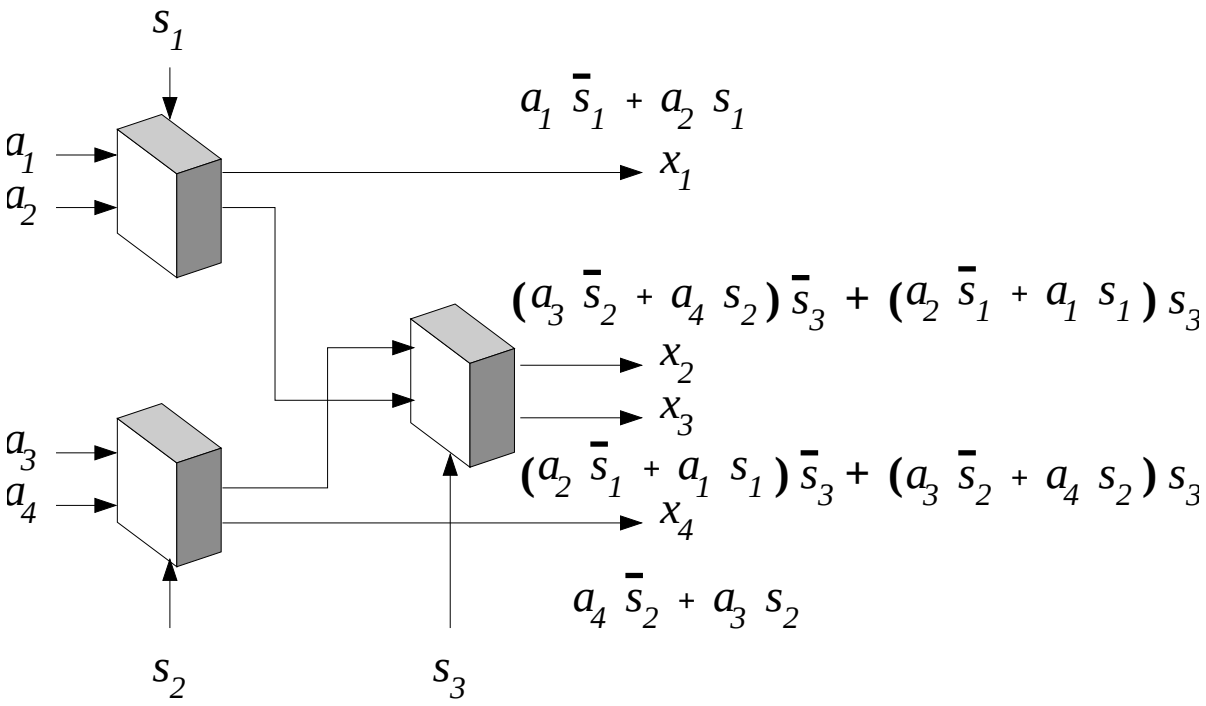
Module Specification



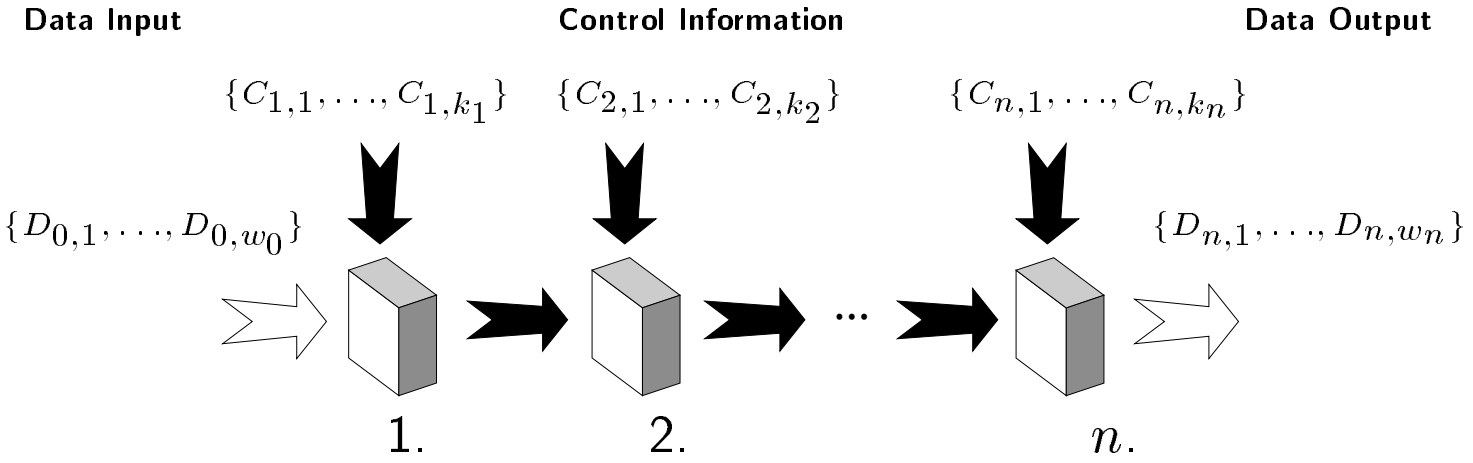
$(s = 0) \Rightarrow x = a, y = b$

$(s = 1) \Rightarrow x = b, y = a$

Interconnect Network



Illegal Transition Checking



Does control information exist to map a setting of logical values on input lines $\{D_{0,1}, D_{0,1}, \ldots, D_{0,w_0}\}$ to a legal set of output values $\{D_{n,1}, D_{n,1}, \ldots, D_{n,w_n}\}$ through n chunks of combinational logic?

For example, if $\{D_{0,1}, D_{0,2}, \ldots, D_{0,w_0}\}$ and $\{D_{n,1}, D_{n,2}, \ldots, D_{n,w_n}\}$ represent an unsecure/illegal state transition, then the propositional problem is to verify that the unsecure/illegal state is unreachable or to give a counter example of how the unsecure/illegal state can be reached from the initial state input.

Unsatisfiable benchmarks, generated in the formal verification of
superscalar and VLIW processors.
==
...

Condition of Availability

The benchmark suite is made available, provided that any publications that
use it will list the reference:
 M.N. Velev, FVP-UNSAT.1.0. Available from: http://www.ece.cmu.edu/~mvelev.
and that the authors of such publications will email
 Miroslav N. Velev (mvelev@ece.cmu.edu) & Randal E. Bryant (Randy.Bryant@cs.cmu.edu)
with the best results achieved, and with enough technical details as to
enable the replication of the experiments.
...

Benchmarks

...
2dlx_ca_mc_ex_bp_f.cnf is the Boolean condition for the correctness of a
dual-issue superscalar version of the above processor with one complete
pipeline and a second pipeline capable of executing only register-register
and register-immediate instructions. The evaluation of the same formula with
BDDs takes 980 seconds of CPU time.

2dlx_cc_mc_ex_bp_f.cnf same as above, but the processor has two complete
pipelines. The evaluation of the same formula with BDDs takes 2,670 seconds
of CPU time.

9vliw_bp_mc.cnf is the Boolean condition for the correctness of a 9-wide
VLIW processor with branch prediction and multicycle functional units [1]
that imitates the Intel Itanium in speculative features such as predicated
execution, register remapping, and advanced loads. The evaluation of the same
formula with BDDs takes 113,640 seconds (31:30 hours) of CPU time.
...
All BDD-based experimrnts were performed on a 336 MHz SUN4.

NOTE:

2dlx_ca_mc_ex_bp_f.cnf, 2dlx_cc_mc_ex_bp_f.cnf, and 9vliw_bp_mc.cnf should be
very challenging for SAT-checkers.

Heuristics

Locally Skewed, Globally Balanced

- Function weights bring inferences up to top
- Combination of weights balances search

Future

- Exploit saved inference information
- Attempt to maximize the number of inferences

Static

- Number of functions with var as input

Lemmas

Memoize and reuse discovered inferences

Typical Results

LBF instance					Performance	
Type	In Vars	Layers	Width	**Solver**	**Time (sec)**	**Backtracks**
Sliding	60	4	6	GRASP	746	32000
				SATO3	510	41800
				SBSAT	51	24000
Random	60	4	6	GRASP	>1000	-
				SATO3	85	11600
				SBSAT	.03	28

Performance of SAT solvers on two LBFs

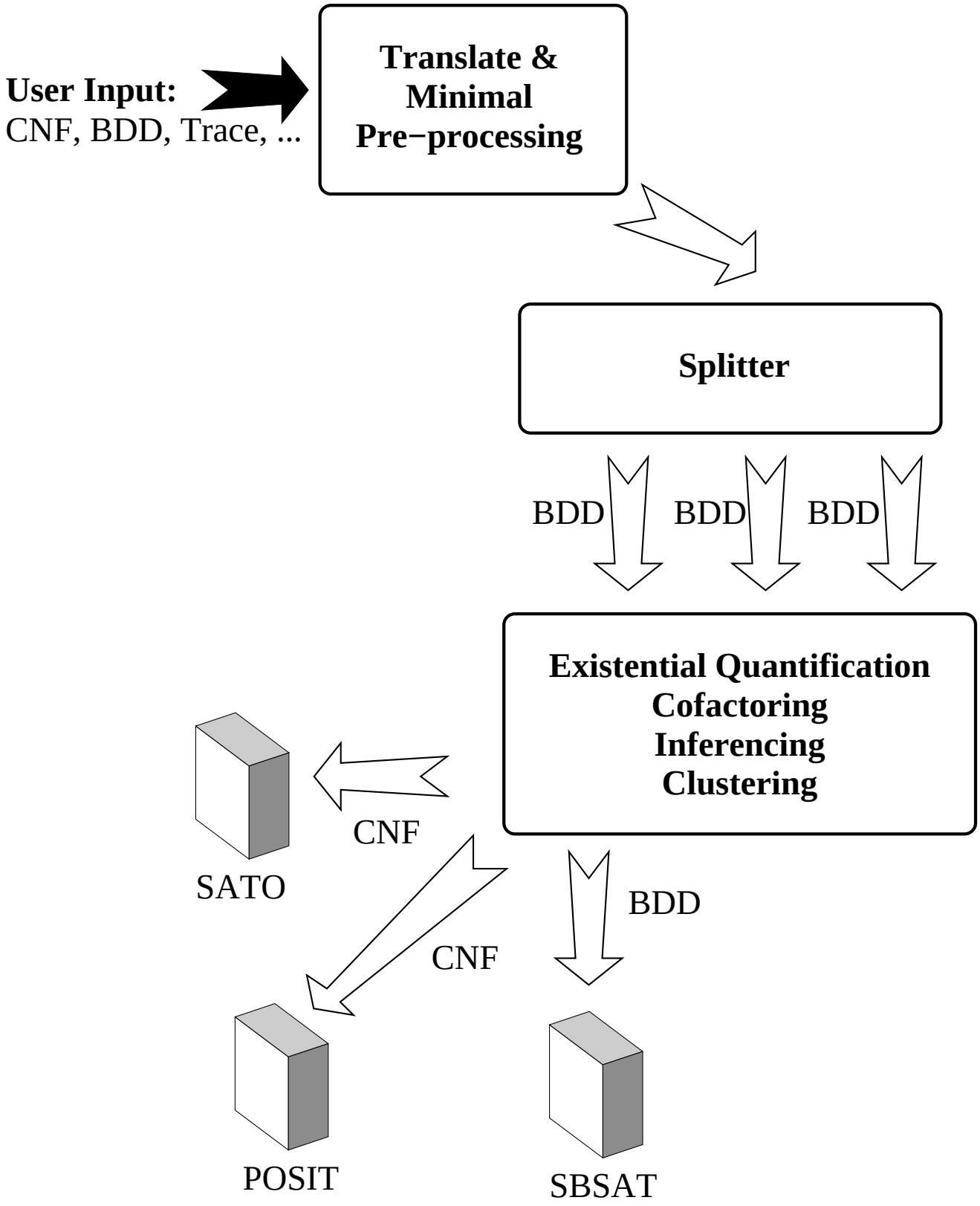
Solver	**Time (sec)**	**Backtracks**
GRASP	>1000	-
SATO3	22	6220
SBSAT	0.17	171

Performance of SAT solvers on ITC formula

Heuristic	**Lemmas**	**Time (sec)**	**Backtracks**
Static	NO	376	617351
Static	YES	17	7847
LSGB (1.3)	NO	0.17	171
LSGB (1.3)	YES	0.16	142

Performance of SBSAT on ITC formula with and without lemmas

User Interface



Translation to CNF

Expression: $v_3 = ite(v_0, v_1, v_2);$

Karnaugh Map:

	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	0	1	1	0
10	1	0	0	1

CNF:

$$(v_0 \vee v_2 \vee \bar{v}_3)$$
$$(v_0 \vee \bar{v}_2 \vee v_3)$$
$$(\bar{v}_0 \vee \bar{v}_1 \vee v_3)$$
$$(\bar{v}_0 \vee v_1 \vee \bar{v}_3)$$

Additional Clauses:

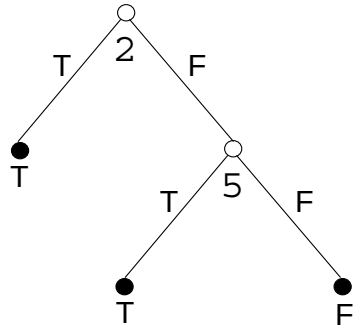
$$(\bar{v}_1 \vee \bar{v}_2 \vee v_3)$$
$$(v_1 \vee v_2 \vee \bar{v}_3)$$

What Heuristics Like:

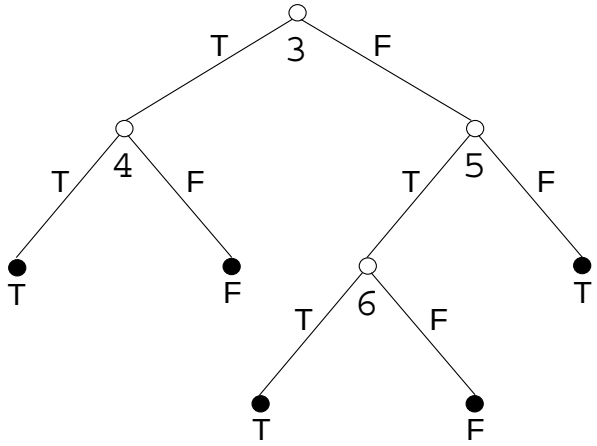
$$(v_0 \wedge v_1) \rightarrow v_3 \qquad (v_0 \wedge v_3) \rightarrow v_1$$
$$(\bar{v}_0 \wedge v_2) \rightarrow v_3 \qquad (\bar{v}_0 \wedge v_3) \rightarrow v_2$$
$$(v_0 \wedge \bar{v}_1) \rightarrow \bar{v}_3 \qquad (v_0 \wedge \bar{v}_3) \rightarrow \bar{v}_1$$
$$(\bar{v}_0 \wedge \bar{v}_1) \rightarrow \bar{v}_3 \qquad (\bar{v}_0 \wedge \bar{v}_3) \rightarrow \bar{v}_2$$
$$(v_1 \wedge v_2) \rightarrow v_3 \qquad (\bar{v}_1 \wedge \bar{v}_2) \rightarrow \bar{v}_3$$
$$(v_1 \wedge \bar{v}_3) \rightarrow \bar{v}_0, \bar{v}_2$$
$$(v_2 \wedge \bar{v}_3) \rightarrow v_0, \bar{v}_1$$
$$(\bar{v}_1 \wedge v_3) \rightarrow v_0, v_2$$
$$(\bar{v}_2 \wedge v_3) \rightarrow v_0, v_1$$

Co-factoring

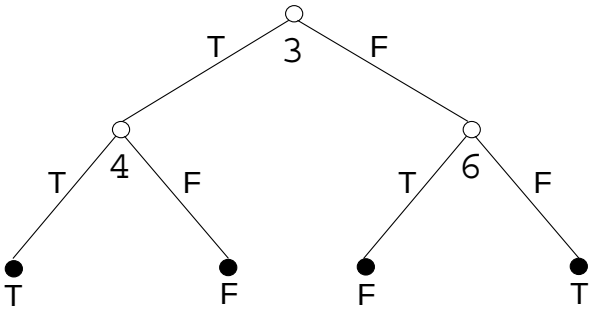
c = and(2, 5);



f = ite(3, 4, and(5, 6));



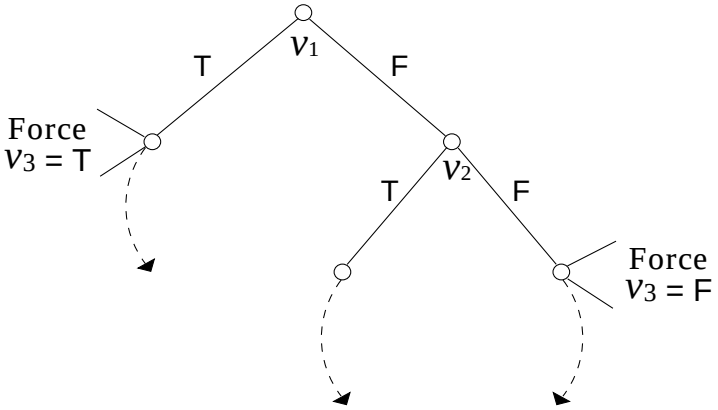
GCF(c, f) = ite(3, 4, 6);



Existential Quantification

Existential Quantification

Expression: $v_3 = ite(v_0, v_1, v_2)$ $\xrightarrow{\exists x\,(v_0)}$ $v_3 = (v_1 \vee v_2)$

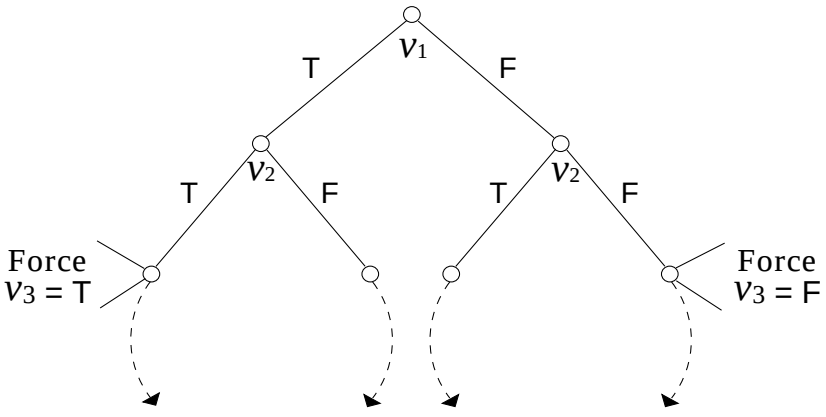


CNF:

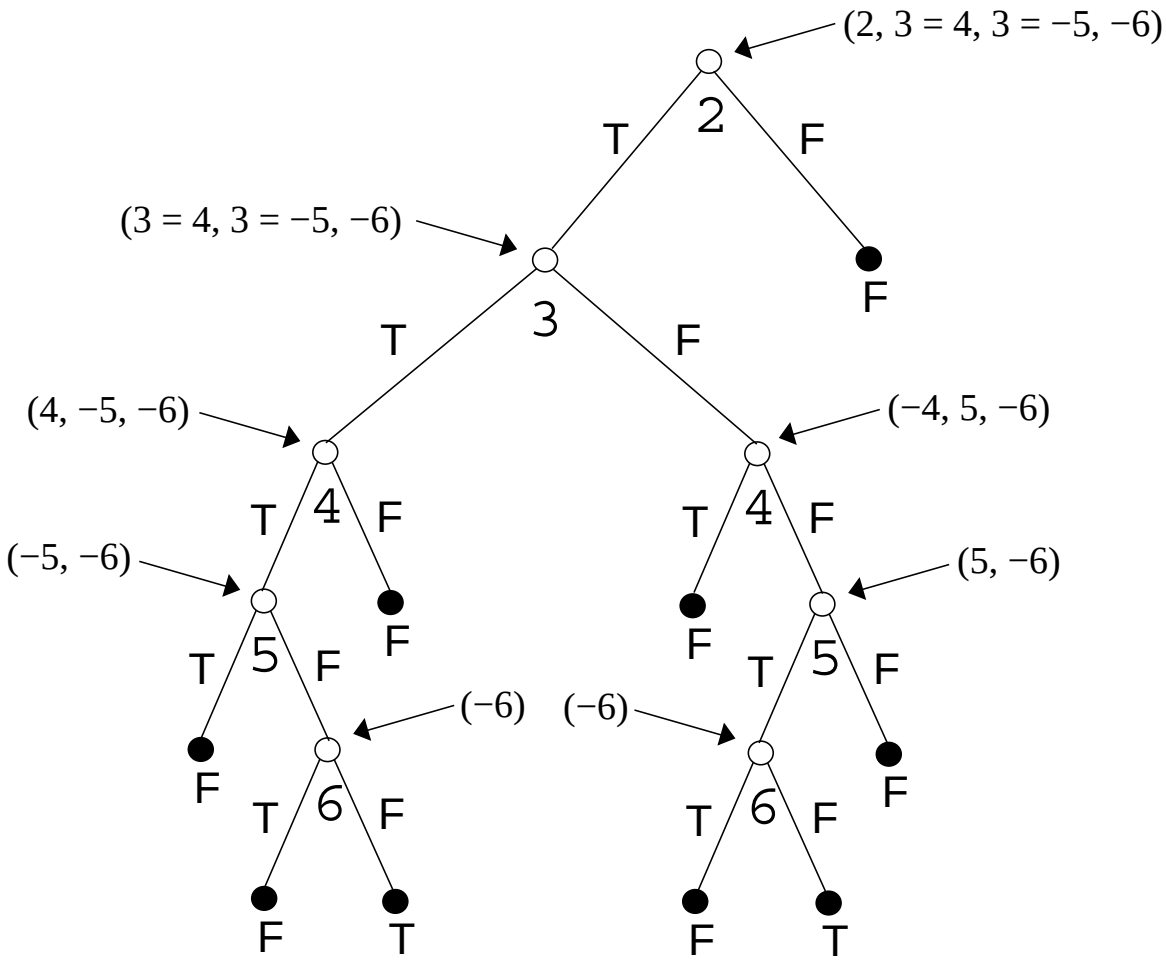
$(v_0 \vee v_2 \vee \bar{v}_3)$
$(v_0 \vee \bar{v}_2 \vee v_3)$
$(\bar{v}_0 \vee \bar{v}_1 \vee v_3)$
$(\bar{v}_0 \vee v_1 \vee \bar{v}_3)$

Resolution

$(v_1 \vee v_2 \vee \bar{v}_3)$
$(\bar{v}_1 \vee \bar{v}_2 \vee v_3)$



Inferences


(2, 3 = 4, 3 = −5, −6)
2
T
F
F
(3 = 4, 3 = −5, −6)
3
T
F
(4, −5, −6)
(−4, 5, −6)
4
T
F
4
T
F
(−5, −6)
(5, −6)
5
T
F
F
F
5
T
F
(−6)
(−6)
F
F
6
T
F
6
T
F
F
T
F
T

CNF PM & Clustering

Example CNF file:

p cnf 5 4
2 4 −5 0
2 −4 5 0
−2 −3 5 0
−2 3 −5 0

Expression: 5 = ite(2, 3, 4)

Example CNF file:

p cnf 6 5
2 −3 4 5 6 0
−2 −6 0
3 −6 0
−4 −6 0
−5 −6 0

Expression: 6 = and(−2, 3, −4, −5)

Example CNF file:

p cnf 6 5
−2 −3 4 5 −6 0
2 6 0
3 6 0
−4 6 0
−5 6 0

Expression: 6 = or(−2, −3, 4, 5)