Computing prime implicants

David Déharbe Pascal Fontaine Daniel Le Berre
Bertrand Mazure

UFRN, Brazil

Inria, U. of Lorraine, France

CRIL-CNRS, U. of Artois, France

FMCAD 2013, October 21, Portland

Model, implicant, prime implicant

Example

- Let $\phi = \{a \vee b, a \vee c, \neg d \vee \neg e \vee \neg f\}$
- $\{a, b, c, d, e, \neg f\}$ is a model of ϕ
- $a \wedge b \wedge c \wedge d \wedge e \wedge \neg f$ and $a \wedge c \wedge \neg d$ are implicants of ϕ
- $a \wedge \neg f$, $b \wedge c \wedge \neg f$ are prime implicants of ϕ

- A model of a formula is an implicant of that formula
- From one implicant, one can derive at least one prime implicant
- SAT solvers compute models
- How to make them compute prime implicants (efficiently) ?

Clauses, cardinality constraints, pseudo-boolean constraints

Various boolean constraints

clauses	$a \vee \neg b \vee c$	
cardinality	$\sum l_i \{\leq, =, \geq\} k$	$a + b + c + d \leq 1$
pseudo boolean	$\sum w_i \times l_i \{\leq, =, \geq\} k$	$4 \times a + 2 \times b + c + d \geq 6$

- Boolean variables seen as 0/1 integer variables
- Normalization : $\neg a + \neg b + \neg c + \neg d \geq 3$
- Clauses are specific cardinality constraints with $k = 1$
  $a \vee \neg b \vee c \equiv a + \neg b + c \geq 1$
- Cardinality constraints are specific PB constraints with $w_i = 1$.

Motivation : SAT-based MAXSAT

Example

- Let $\phi = \{\neg a \vee b, \neg a \vee c, a, \neg b, a \vee b, \neg c \vee b\}$
- MAXSAT (ϕ) = minimize $\sum s_i$ such that ϕ' with $\phi' = \{s_1 \vee \neg a \vee b, s_2 \vee \neg a \vee c, s_3 \vee a, s_4 \vee \neg b, s_5 \vee a \vee b, s_6 \vee \neg c \vee b\}$
- $S = \{s_i\}$ called "selector variables"

- Use SAT solver to find models M of $\phi' \wedge (\sum s_i < k)$ with decreasing $k = |M \cap S|$ until formula inconsistent.
- $\sum s_i < k$ being either a native cardinality constraints (Sat4j) or translated into CNF (QMaxSat).
- $\sum w_i \times s_i < k$ (pseudo-boolean constraint) for Weighted [Partial] MaxSat

Motivation : SAT-based MAXSAT

Example

- Let $\phi = \{\neg a \vee b, \neg a \vee c, a, \neg b, a \vee b, \neg c \vee b\}$
- MAXSAT (ϕ) = minimize $\sum s_i$ such that ϕ' with
  $\phi' = \{s_1 \vee \neg a \vee b, s_2 \vee \neg a \vee c, s_3 \vee a, s_4 \vee \neg b, s_5 \vee a \vee b, s_6 \vee \neg c \vee b\}$
- $S = \{s_i\}$ called "selector variables"

- Use SAT solver to find models M of $\phi' \wedge (\sum s_i < k)$ with decreasing $k = |M \cap S|$ until formula inconsistent.
- $\sum s_i < k$ being either a native cardinality constraints (Sat4j) or translated into CNF (QMaxSat).
- $\sum w_i \times s_i < k$ (pseudo-boolean constraint) for Weighted [Partial] MaxSat
- if $M = \{a, b, c, s_1, s_2, s_3, s_4, \neg s_5, \neg s_6\}$, $k = 4$
  The bound is not tight ! s_1, s_2, s_3 are satisfied while their corresponding clauses are satisfied !

Improve upper bound for SAT-based MAXSAT solvers

- Two possible approaches :
  - Change the encoding : equivalence instead of implication for selector variables
    $\neg s_i \rightarrow c_i$ becomes $\neg s_i \leftrightarrow c_i$
    adds $|\phi|$ binary clauses to ϕ'
  - Use a prime implicant instead of a model to compute the upper bound
- Requirements :
  - fast : computation need to be done at each model found
  - compatible with incremental SAT (no/small data structure overhead)
  - should work with clauses, cardinality constraints, and pseudo-boolean constraints

Abstract computation of prime implicants

procedure $\textsc{Prime}(\mathcal{C}, M_0, \Pi_0)$
    $M, \Pi \leftarrow M_0, \Pi_0$
    while $\ell \in M \setminus \Pi$ **do**
        if $\mathit{Required}(M, \ell, \mathcal{C})$ **then** $\Pi \leftarrow \Pi \cup \{\ell\}$
        else $M \leftarrow M \setminus \{\ell\}$
    return Π

- M_0 is the model returned by the SAT solver
- *Required*() checks if a given literal ℓ is required in the implicant, i.e. $\exists c \in \mathcal{C}$ such that satisfying ℓ is mandatory to satysfy c [Castell96].
- Π_0 easy to find required literals (e.g. propagated literals).
- In practice, $|M_0 \setminus \Pi_0| << |\Pi_0|$
- Works for any kind of constraints
- Needs to be refined for efficient implementation !

Prime implicants for clauses (counter based)

procedure $\textsc{Prime}(\mathcal{C}, M_0, \Pi_0)$

  $M, \Pi \leftarrow M_0, \Pi_0$

  for all $\ell \in M$ **do** $W(\ell) \leftarrow \emptyset$

  for all $c \in \mathcal{C}$ **do**

    $\mathrm{N}[c] \leftarrow 0$

    for all $\ell \in c$ **do** $W(\ell) \leftarrow W(\ell) \cup \{c\}$

  for all $\ell \in M$ **do**

    for all $c \in W(\ell)$ **do** $\mathrm{N}[c] \leftarrow \mathrm{N}[c] + 1$

  for all $\ell \in M \setminus \Pi$ **do**

    if $\exists c \in W(\ell)\ .\ \mathrm{N}[c] = 1$ **then**

      $\Pi \leftarrow \Pi \cup \{\ell\}$

    else

      for all $c \in W(\ell)$ **do** $\mathrm{N}[c] \leftarrow \mathrm{N}[c] - 1$

      $M \leftarrow M \setminus \{\ell\}$

  return Π

Prime implicants for cardinality constraints (counter based)

procedure $\textsc{Prime}(\mathcal{C}, M_0, \Pi_0)$
    $M, \Pi \leftarrow M_0, \Pi_0$
    for all $\ell \in M$ **do** $W(\ell) \leftarrow \emptyset$
    for all $c \in \mathcal{C}$ **do**
        $\mathrm{N}[c] \leftarrow 0$
        for all $\ell \in c$ **do** $W(\ell) \leftarrow W(\ell) \cup \{c\}$
    for all $\ell \in M$ **do**
        for all $c \in W(\ell)$ **do** $\mathrm{N}[c] \leftarrow \mathrm{N}[c] + 1$
    for all $\ell \in M \setminus \Pi$ **do**
        if $\exists c \in W(\ell)\ .\ \mathrm{N}[c] = \mathit{degree}(c)$ **then**
            $\Pi \leftarrow \Pi \cup \{\ell\}$
        else
            for all $c \in W(\ell)$ **do** $\mathrm{N}[c] \leftarrow \mathrm{N}[c] - 1$
            $M \leftarrow M \setminus \{\ell\}$
    return Π

About Counter-based approaches

- Complexity is linear in the size of $\mathcal{C}$: $\mathcal{O}(\sum_{c \in \mathcal{C}} |c|)$
- Works for both clauses and cardinality constraints
- Easy to implement outside the solver
- What about early detection of required literals ?
- What about pseudo boolean constraints ?
- What about incremental SAT solving ?

Abstract propagation-based algorithm

1: **procedure** $\textsc{Prime}(\mathcal{C}, M_0, \Pi_0)$

2: $\quad M, \Pi \leftarrow M_0, \Pi_0$

3: $\quad \Pi \leftarrow \Pi \cup \textsc{Implied}(\mathcal{C}, M)$ ▷ Propagates required literals

4: $\quad$ **while** $\ell \in M \setminus \Pi$ **do**

5: $\quad\quad M \leftarrow M \setminus \{\ell\}$

6: $\quad\quad \Pi \leftarrow \Pi \cup \textsc{Implied}(\mathcal{C}, M)$ ▷ Propagates removal of ℓ

7: $\quad$ **return** Π

- $\textsc{Implied}()$ propagates truth value similarly to Unit Propagation
- New invariant : each literal picked up at line 4 is not required
- We can reuse here the data structures found in modern SAT solvers !

informatics mathematics

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

UNIVERSITÉ D'ARTOIS

Prime implicants using watched literals

procedure $\textsc{Prime}(\mathcal{C}, M_0, \Pi_0, W)$
  $M, \Pi \leftarrow M_0, \Pi_0$
  for all $\ell \in M \setminus \Pi$ **do** ▷ Watch satisfied literals
    $\textsc{Implied}_W(\mathcal{C}, M, \bar{\ell}, \Pi, W)$
  while $\ell \in M \setminus \Pi$ **do**
    $M \leftarrow M \setminus \{\ell\}$
    $\textsc{Implied}_W(\mathcal{C}, M, \ell, \Pi, W)$ ▷ Propagates removal of ℓ
  return Π

procedure $\textsc{Implied}_W(\mathcal{C}, M, \ell, \textbf{ref } \Pi, \textbf{ref } W)$
  $W_\ell \leftarrow W(\ell)$
  for all $c \in W_\ell$ **do**
    $\textsc{Hdl\_constr}(c, M, \ell, \Pi, W)$ ▷ Specific to each c

$W(\ell)$ = constraints "watched" for literal ℓ

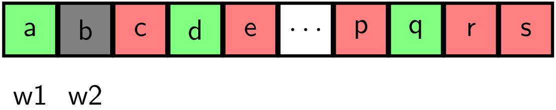
HDL_CONSTR for clause or cardinality constraints

procedure HDL_CONSTR($c, M, \ell,$ **ref** Π, **ref** W)
  if $\exists \ell' \in c \cap M \,.\, \ell' \notin W^{-1}(c)$ **then**
    $W \leftarrow (W \cup \{\ell' \mapsto c\}) \setminus \{\ell \mapsto c\}$
  else $\Pi \leftarrow \Pi \cup (W^{-1}(c) \setminus \{\ell\})$

- $W^{-1}(c)$ = literals "watched" in constraint c
- Just like lazy data structure management during unit propagation (in clauses or cardinality constraints)
- Watches satisfied literals : there is at least one such literal per clause.
- One important difference : constraints are traversed only once. that condition must hold to achieve linear time !

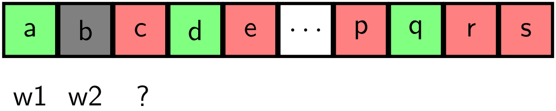
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



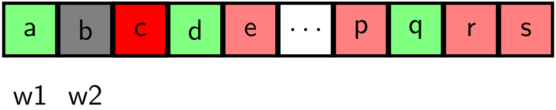
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



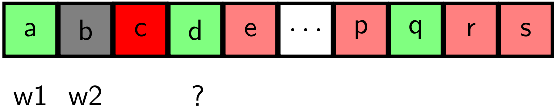
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



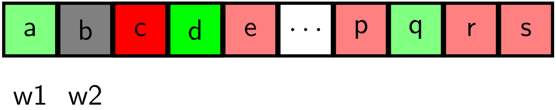
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



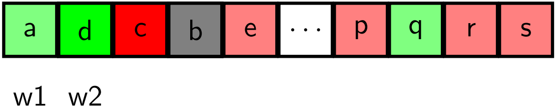
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



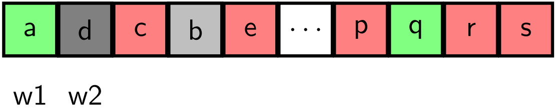
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



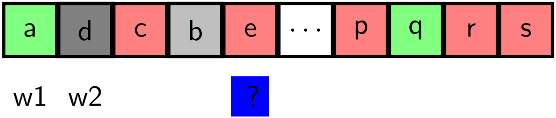
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



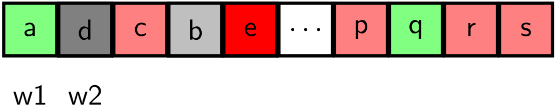
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



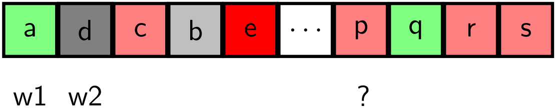
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



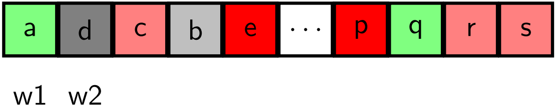
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



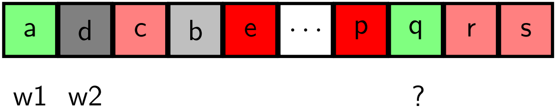
informatics mathematics

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

UNIVERSITÉ D'ARTOIS

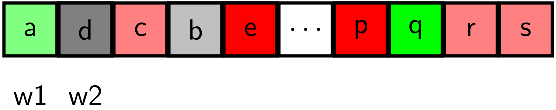
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



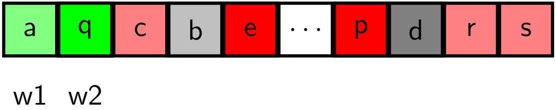
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



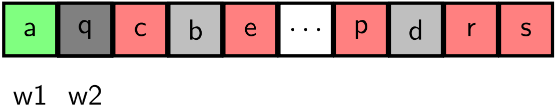
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



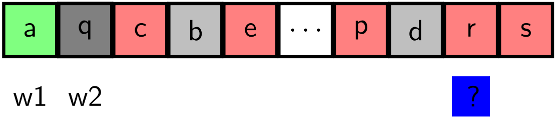
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



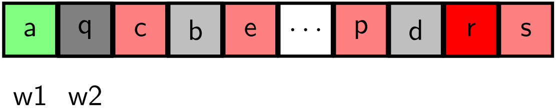
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



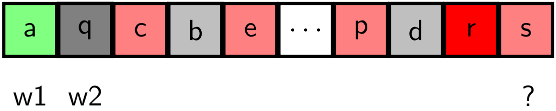
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



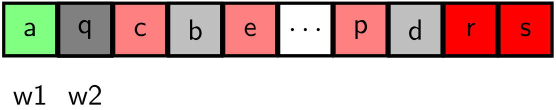
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



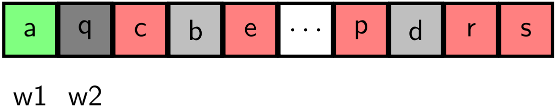
Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



Prime Implicant specific propagation

- Triggered when a literal is removed from M
- Procedure looks for a satisfied literal
- Some literals may be deleted
- Propagates a required literal



a is mandatory/required !

HDL_CONSTR for arbitrary constraints

procedure HDL_CONSTR($c, M, \ell,$ **ref** $\Pi,$ **ref** W)

    $\Pi \leftarrow \Pi \cup \{\ell' \in W^{-1}(c) \mid \mathit{Required}(M, \ell', c)\}$

    if $\Pi \not\models c$ **then**

        Choose W' **such that**

            $W' \subseteq (W^{-1}(c) \cup M) \setminus \{\ell\}$

            $(\Pi \cup W') \cap M \models c$

            $\forall \ell' \in W' \setminus \Pi \,.\, \neg \mathit{Required}(W' \cup \Pi, \ell', c)$

        in $W^{-1}(c) \leftarrow W'$

PB constraints can propagate truth values without being satisfied :

$4 \times a + 2 \times b + c + d \geq 6$ propagates a.

Experimental results : some Safarpour *et al* benchmarks

#vars (M)	#cla (M)	#literals (M)	#implied (M)	Counters (s)	Watched (s)
2.3	1.7	4.0	0.5	4.842	**0.736**
1.5	1.1	2.7	0.4	2.860	**0.495**
2.0	1.5	3.9	0.5	4.191	**0.486**
1.6	1.2	2.9	0.4	3.956	**0.377**
1.8	1.0	2.8	0.3	4.008	**0.354**
2.0	1.6	4.5	0.4	2.567	**0.486**
2.0	1.6	4.6	0.4	2.493	**0.493**
2.2	1.7	4.8	0.4	9.225	**0.510**
2.2	1.7	4.8	0.4	8.946	**0.490**
2.2	1.7	4.8	0.4	6.086	**0.556**
1.5	1.2	3.4	0.3	4.250	**0.366**
1.5	1.2	3.4	0.3	4.172	**0.370**

Experimental results : MAXSAT 10 benchmarks

Sat4j MaxSat 2.3.6, 2GB of memory, 1200s timeout

	MAXSAT 544 (77)	Partial MS 1122 (497)	Weighted MS 349 (-)	WPMS 660 (132)
models $\rightarrow$	10 (8)	485 (269)	59	211 (36)
models $\leftrightarrow$	7 (4)	491 (270)	65	211 (35)
PI counters	5 (3)	487 (268)	-	-
PI this work	10 (8)	490 (269)	61	215 (38)

Conclusion

- Prime implicant computation almost for free in CDCL solvers : no computational nor memory overhead
- Works for different kind of constraints.
- Linear behavior depends on the kind of constraints (i.e. guaranteed to be traversed only once during propagation)
- All presented algorithms properly implemented as separate classes in Sat4j 2.3.6 (to be released)
- For MaxSat, few important selector variables removed : might need to consider specific heuristics for that.
- How to enumerate all prime implicants of a formula ?