

# Towards an Optimal CNF Encoding of Boolean Cardinality Constraints

---

**Carsten Sinz**

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



# Boolean Cardinality Constraints

- Number restrictions: "at most", "at least"

- Notation:

$\leq k(x_1, \dots, x_n)$  : at most  $k$  true

$\geq k(x_1, \dots, x_n)$  : at least  $k$  true

- Examples:

$$\geq 3(x, y, z) = x \wedge y \wedge z$$

$$\leq 1(x, y, z) = \neg(x \wedge y) \wedge \neg(x \wedge z) \wedge \neg(y \wedge z)$$

$$= (\neg x \vee \neg y) \wedge (\neg x \vee \neg z) \wedge (\neg y \vee \neg z)$$

# Why Translate Cardinality Constraints to CNF?

---

- Cardinality constraints occur frequently in practice:
  - product configuration, (frequency) assignment problems, discrete tomography,...
- SAT-Solvers (typically) can handle formulae in CNF only
- SAT-Solving has seen tremendous progress over the last years
- Many high-performance implementations of SAT-Solvers are available

# Standard (Naïve) Encoding

$\leq k(x_1, \dots, x_n)$  is translated to

$$\bigwedge_{\substack{M \subseteq \{1, \dots, n\} \\ |M| = k+1}} \bigvee_{i \in M} \neg x_i .$$

This results in  $\binom{n}{k+1}$  clauses of length  $k+1$ .

Worst case ( $k = \lceil n/2 \rceil + 1$ ):  $O(2^n / \sqrt{n/2})$ ,

i.e. exponential number of clauses

# Encoding Example:

## $\leq 1(x_1, \dots, x_{100})$

Naïve Encoding:

$$\begin{aligned} & (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_5) \wedge \\ & \underbrace{(\neg x_1 \vee \neg x_5) \wedge \dots \wedge (\neg x_{98} \vee \neg x_{99}) \wedge (\neg x_{98} \vee \neg x_{100}) \wedge (\neg x_{99} \vee \neg x_{100})}_{\binom{100}{2} = 4950 \text{ clauses}} \end{aligned}$$

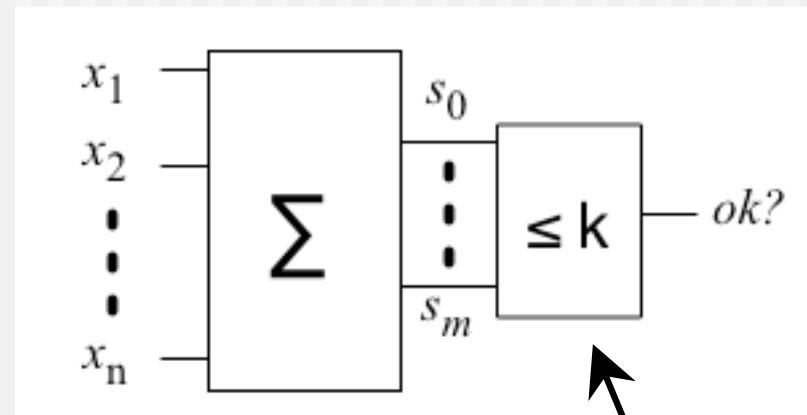
New Encoding 1 ( $LT_{SEQ}$ ): 296 clauses

New Encoding 2 ( $LT_{PAR}$ ): 669 clauses

[  $\leq 10(x_1, \dots, x_{1000})$ : 312176429322332191039274595287040000000 /  
20969 / 6950 ]

# New Encodings: Based on Hardware Counters

- General approach: *counting and comparing*



counter  
(unary or binary,  
sequential or parallel)

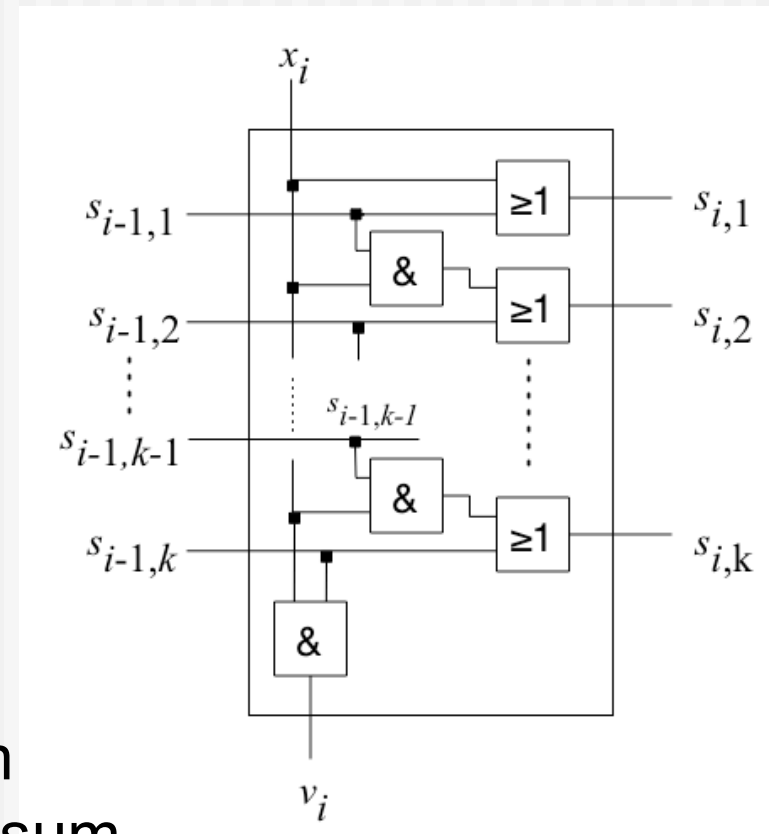
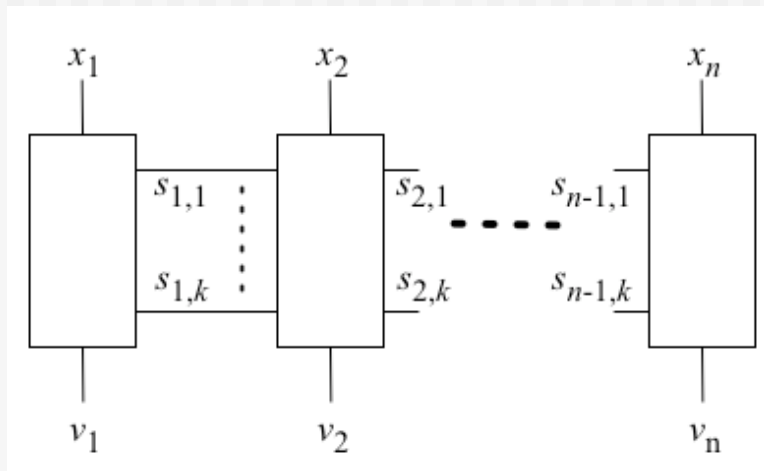
comparator

# Classification of Counters

---

- **Unary:** (intermediate) results are represented as unary numbers (e.g.,  $5 \cong \text{IIII}$ )
- **Binary:** (intermediate) results are represented as binary numbers (e.g.  $5 \cong 101$ )
- **Sequential:** inputs are summed up one after the other
- **Parallel:** inputs are summed up concurrently

# Encoding 1 ( $LT_{SEQ}$ ): Sequential Unary Counter



$s_{i,j}$ :  $j$ -th digit of  $i$ -th partial sum  
 $v_i$ : overflow bit for  $k$ -th partial sum

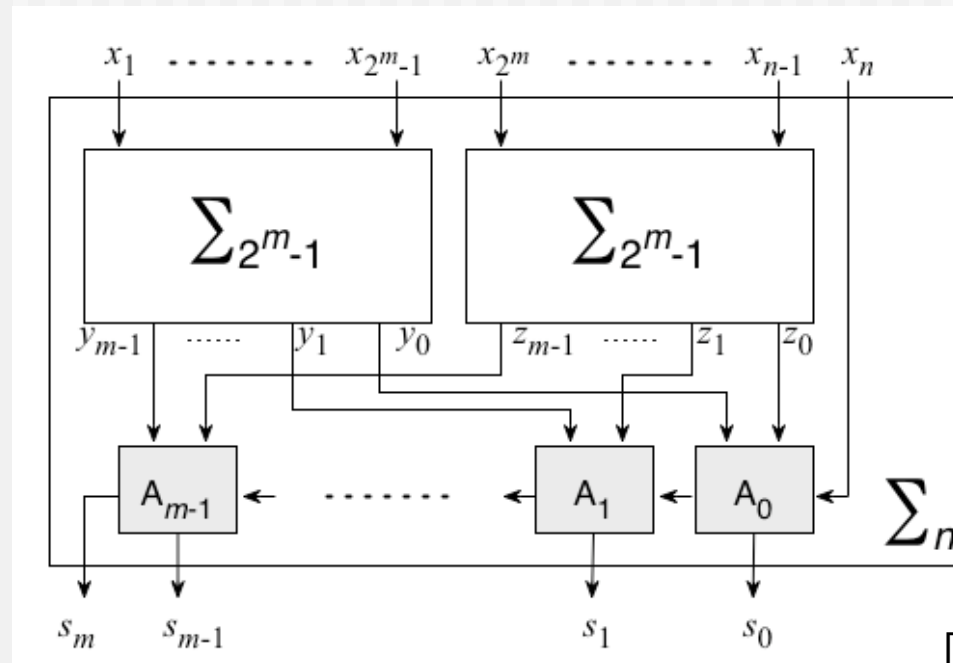
# Encoding 1: Conversion to CNF

$$\begin{array}{l}
 (\neg x_1 \vee s_{1,1}) \\
 (\neg s_{1,j}) \quad \text{for } 1 < j \leq k \\
 (\neg x_i \vee s_{i,1}) \\
 (\neg s_{i-1,1} \vee s_{i,1}) \\
 (\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}) \\
 (\neg s_{i-1,j} \vee s_{i,j}) \\
 (\neg x_i \vee \neg s_{i-1,k}) \\
 (\neg x_n \vee \neg s_{n-1,k})
 \end{array}
 \left. \vphantom{\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \end{array}} \right\} \text{for } 1 < j \leq k
 \left. \vphantom{\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \end{array}} \right\} \text{for } 1 < i < n$$

$2nk + n - 3k - 1$  clauses

$k(n-1)$  auxiliary (encoding) variables

# Encoding 2 ( $LT_{PAR}$ ): Parallel Binary Counter



[Muller & Preparata, 1975]

(only counter; binary comparator circuit not shown)

$\Sigma_n$ : parallel counter summing up  $n$  inputs  
 $A_i$ : 1-bit-(full-)adders

# Encoding 2: Conversion to CNF

Counter circuit (full adders, half adders not shown):

$$\begin{array}{lll}
 (a \vee b \vee \neg c \vee s_{\text{out}}) & (\neg a \vee b \vee c \vee s_{\text{out}}) & (\neg a \vee \neg b \vee c_{\text{out}}) \\
 (a \vee \neg b \vee c \vee s_{\text{out}}) & (\neg a \vee \neg b \vee \neg c \vee s_{\text{out}}) & (\neg a \vee \neg c \vee c_{\text{out}}) \\
 & & (\neg b \vee \neg c \vee c_{\text{out}})
 \end{array}$$

Comparator circuit (recursive def.):

$$L(k_0) = \begin{cases} \{\{\neg s_0\}\} & \text{if } k_0 = 0 \\ \emptyset & \text{if } k_0 = 1 \end{cases}$$

$$L(k_i \dots k_0) = \begin{cases} \{\{\neg s_i\}\} \cup L(k_{i-1} \dots k_0) & \text{if } k_i = 0 \\ \{\{\neg s_i\}\} \otimes L(k_{i-1} \dots k_0) & \text{if } k_i = 1 \end{cases}$$

$(k_m \dots k_0)$ : binary representation of limit value  $k$

$\otimes$ : clause distribution ( $A \otimes B := \{x \cup y \mid x \in A, y \in B\}$  for clause sets  $A, B$ )

# Encoding 2: Conversion to CNF (cont'd)

Encoding requires:

- $n - \lfloor \log n \rfloor - 1$  full adders,
- at most  $\lfloor \log n \rfloor$  half adders, and
- at most  $\lfloor \log n \rfloor + 1$  clauses for the comparator.

This results in

$$\begin{aligned} & 7(n - \lfloor \log n \rfloor - 1) + 3\lfloor \log n \rfloor + \lfloor \log n \rfloor + 1 \\ & = 7n - 3\lfloor \log n \rfloor - 6 \end{aligned}$$

clauses for the parallel binary counter circuit and

$2n - 2$  additional encoding variables.

# Comparing Sizes of Different Encodings

Encoding	#clauses	#aux. vars	decided
Naïve	$\binom{n}{k+1}$	0	immediately
Sequential unary counter ( $LT_{SEQ}^{n,k}$ )	$\mathcal{O}(n \cdot k)$	$\mathcal{O}(n \cdot k)$	by unit prop.
Parallel binary counter ( $LT_{PAR}^{n,k}$ )	$7n - 3\lfloor \log n \rfloor - 6$	$2n - 2$	by search
Bailleux & Boufkhad [3]	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot \log n)$	by unit prop.
Warners [4]	$8n$	$2n$	by search

- [3] Bailleux, O. and Boufkhad, Y.: Efficient CNF Encoding of Boolean Cardinality Constraints, CP'2003
- [4] Warners, J.P.: A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form. *Inf. Proc. Lett.* **68** (1998)

# Availability & Future Work

---

- Generators for both Encodings (in C++) are available from

`http://www-sr.informatik.uni-tuebingen.de/  
~sinz/CardConstraints`

- To Do: Experimental evaluation
  - Compare different encodings w.r.t. practical applicability

# Conclusion

---

- Two new clausal encodings for Boolean cardinality constraints:
  1. **Encoding 1 [seq./unary]**: Good for small values of  $k$ , decided by unit propagation.
  2. **Encoding 2 [par./binary]**: Encoding with least known number of clauses.
- Further (theoretical) result: no encoding with fewer than  $n$  clauses possible.