On the Reification of Global Constraints

Nicolas Beldiceanu*, Mats Carlsson+,
Pierre Flener^ and Justin Pearson^

*TASC team (INRIA/CNRS), Mines de Nantes
+SICS, P.O. Box 1263, SE-164 29 Kista, Sweden
^Uppsala Univ., Dept. of Info. Technology, Uppsala, Sweden

What is it all about ?

Associate to a constraint $Ctr(Arg_1,Arg_2,\ldots,Arg_n)$ a 0/1 variable B so that:

$Ctr(Arg_1,Arg_2,\ldots,Arg_n) \Leftrightarrow B$

and of course use variable B

*in **other** constraints*

Overview

- **Introduction and motivation**
- How to derive reified global constraints
- Reification of core global constraints
- Categories used in reifying global constraints
- A classification of g.c. wrt. reification
- Conclusion

Reification today

- Easy for arithmetic constraints
- But **not so easy** for global constraints
- Useful for expressing :
  - **Negation** of a constraint
  - **Disjunction** between constraints
  - **Cardinality** operator
- Many kernels implementors and users like it

Motivations

- **Proving the equivalence** of two constraints models (*see PhD thesis of Nadjib Lazaar where one needs to negate global constraints*)

- **Learning models** from positive (and **negative** samples) <= ***our motivation***

Our contribution

*A **simple way** to **look at** global constraints that allows to get reification in an easy way*

- Introduction and motivation
- **How to derive reified global constraints**
- Reification of core global constraints
- Categories used in reifying global constraints
- A classification of g.c. wrt. reification
- Conclusion

Defining a global constraint

- A global constraint GC(A) is defined by :
  - Some **restrictions** R(A),
  - Some **condition** C(A).

- Example :

Constraint	global_contiguity(VARIABLES)
Argument	VARIABLES : collection(var−dvar)
Restrictions	required(VARIABLES, var) VARIABLES.var ≥ 0 VARIABLES.var ≤ 1

condition

Enforce all variables of the VARIABLES collection to be assigned value 0 or 1. In addition, all variables assigned to value 1 appear contiguously.

Defining a global constraint

- Why to **distinguish** between restriction and condition?
  Because restrictions ***also apply to the negation***

Defining a global constraint

- Why to **distinguish** between restriction and condition?
  Because restrictions ***also apply to the negation***

- Remark

  All constraint solvers (and the catalog) made a mistake while defining automata constraints :
  not providing explictly the alphabet

  => you ***cannot*** *negate the constraint !*

Reification of global constraint

$$GC(\mathcal{A}) \qquad R(\mathcal{A}) \wedge C(\mathcal{A})$$

$$R(\mathcal{A}) \wedge (C(\mathcal{A}) \Leftrightarrow b)$$

Intuition for deriving reified g.c.

Defining the meaning of most global constraints obeys a **determine** and **test** scheme:

1) **Determine** ***(never fails !, PFD)***

   computes values from some arguments

2) **Test**

   simple test on the computed values
   (*e.g., arithmetic, automaton*)

Intuition for deriving reified g.c. (*alldifferent*)

1) Determine

sort *the original variables*

2) Test

check that they are strictly increasing

Intuition for deriving reified g.c. (*cumulative*)

1) Determine

compute the cumulated resource consumption at each task start (since the resource consumption increases at these points)

2) Test

check that all previously computed resource consumption are less than or equal to the overall limit

Intuition for deriving reified g.c. (*nvalue*)

1) Determine

compute the number of distinct values

2) Test

check that the computed number of distinct values is actually equal to the target

Intuition for deriving reified g.c. (*cycle*)

1) Determine

sort the successor variables (as for alldifferent)

compute the number of cycles

2) Test

check that sorted values are strictly increasing

check that number of cycle is equal to the target

Pure Functional Dependency Constraint

No additional condition is imposed by the constraint other than **determining** some of its variables

*It can therefore **never fail** when the variables to be determined are unrestricted !*

Pure Functional Dependency Constraint (*examples*)

```
element(INDEX, TABLE, VALUE)
```

```
INDEX  :  dvar
TABLE  :  collection(value−dvar)
VALUE  :  dvar
```

EXAMPLE

$$(3, \langle 6, 9, 2, 9 \rangle, 2)$$

- Functional dependency: VALUE determined by INDEX and TABLE.

Pure Functional Dependency Constraint (*examples*)

```
global_cardinality(VARIABLES, VALUES)
```

```
VARIABLES  :  collection(var−dvar)
VALUES     :  collection(val−int, noccurrence−dvar)
```

EXAMPLE

$$\left(\begin{array}{l} \langle 3,3,8,6 \rangle , \\ \left\langle \begin{array}{ll} \mathtt{val}-3 & \mathtt{noccurrence}-2, \\ \mathtt{val}-5 & \mathtt{noccurrence}-0, \\ \mathtt{val}-6 & \mathtt{noccurrence}-1 \end{array} \right\rangle \end{array} \right)$$

- Functional dependency: VALUES.noccurrence determined by VARIABLES and VALUES.val.

Pure Functional Dependency Constraint (*examples*)

`nvalue(NVAL, VARIABLES)`

```
NVAL       :  dvar
VARIABLES  :  collection(var−dvar)
```

EXAMPLE

$(4, \langle 3, 1, 7, 1, 6 \rangle)$

- Functional dependency: NVAL determined by VARIABLES.

Pure Functional Dependency Constraint (*examples*)

`sort(VARIABLES1, VARIABLES2)`

`VARIABLES1 : collection(var−dvar)`
`VARIABLES2 : collection(var−dvar)`

EXAMPLE

$$\left(\begin{array}{c} \text{var}-1, \\ \left\langle \begin{array}{c} \text{var}-9, \\ \text{var}-1, \\ \text{var}-5, \end{array} \right\rangle, \\ \text{var}-2, \\ \text{var}-1 \\ \text{var}-1, \\ \left\langle \begin{array}{c} \text{var}-1, \\ \text{var}-1, \\ \text{var}-2, \end{array} \right\rangle \\ \text{var}-5, \\ \text{var}-9 \end{array}\right)$$

Functional dependency: VARIABLES2 determined by VARIABLES1.

Pure Functional Dependency Constraint (*keyword in the catalogue*)

3.7.197 Pure functional dependency [90 CONS]

- abs_value,
- among,
- among_diff_0,
- among_interval,
- among_modulo,
- eq_cst,
- equivalent,
- exactly,
- gcd,
- global_cardinality,

23% of the constraints of the catalogue

Reification of g.c.

RESTRICTIONS **DETERMINE** **CHECK**

$$\boxed{R(\mathcal{A})} \wedge \boxed{CF_1(\mathcal{A}_1, \mathcal{V}_1) \wedge \cdots \wedge CF_p(\mathcal{A}_p, \mathcal{V}_p)} \wedge \boxed{(CN(\mathcal{A}_{p+1}) \Leftrightarrow b)}$$

pure functional dependency ctrs

arith. or automaton ctrs

- $\mathcal{V}_i$ (with $1 \leq i \leq p$) is a non-empty set of distinct unrestricted variables, i.e., it has an empty intersection with $\mathcal{A} \cup \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_{i-1} \cup \mathcal{V}_{i+1} \cup \cdots \cup \mathcal{V}_p$.
- $\mathcal{A}_i \subseteq \mathcal{A} \cup \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_{i-1}$ (with $1 \leq i \leq p$), i.e., $\mathcal{A}_i$ gets fixed when $\mathcal{A}, \mathcal{V}_1, \ldots, \mathcal{V}_{i-1}$ are fixed.
- $\mathcal{A}_{p+1}$ has a non-empty intersection with $\mathcal{V}_1 \cup \cdots \cup \mathcal{V}_p$ and is included in $\mathcal{A} \cup \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_p$.
- $\mathcal{V}_i$ has a non-empty intersection with $\mathcal{A}_{i+1} \cup \cdots \cup \mathcal{A}_{p+1}$, i.e., each introduced variable is used at least once.

- Introduction and motivation
- How to derive reified global constraints
- **Reification of core global constraints**
- Categories used in reifying global constraints
- A classification of g.c. wrt. reification
- Conclusion

Core global constraints

3.7.65 Core [11 CONS]

- alldifferent,
- cumulative,
- cycle,
- diffn,
- disjunctive,
- element (see also elem for the usage),
- global_cardinality,
- global_cardinality_with_costs,
- minimum_weight_alldifferent,
- nvalue,
- sort. **(reformulation)**

$$\text{ALLDIFFERENT}(\langle v_1, \ldots, v_n \rangle)$$

PFD

$$\text{SORT}(\langle v_1, \ldots, v_n \rangle, \langle w_1, \ldots, w_n \rangle) \wedge$$

CHECK

$$(w_1 < w_2 \wedge \cdots \wedge w_{n-1} < w_n) \Leftrightarrow b$$

$$\text{GLOBAL\_CARDINALITY}(\langle x_1, \ldots, x_n \rangle, \langle v_1\ o_1, \ldots, v_m\ o_m \rangle)$$

PFD

$$\text{GLOBAL\_CARDINALITY}(\langle x_1, \ldots, x_n \rangle, \langle v_1\ p_1, \ldots, v_m\ p_m \rangle) \wedge$$

CHECK

$$(o_1 = p_1 \wedge \cdots \wedge o_m = p_m) \Leftrightarrow b$$

$$\text{ELEMENT}(i, \langle t_1, \ldots, t_n \rangle, v)$$

PFD

$$\text{ELEMENT}(i, \langle t_1, \ldots, t_n \rangle, w) \wedge$$

CHECK

$$(v = w) \Leftrightarrow b$$

$$\text{CUMULATIVE}(\langle s_1\ d_1\ e_1\ r_1, \ldots, s_n\ d_n\ e_n\ r_n\rangle, limit)$$

PFD

For each pair of tasks i, j

- For $j = i$: $(d_i = 0 \wedge r_{ij} = 0) \vee (d_i > 0 \wedge r_{ij} = r_i)$
- For $j \neq i$: $((s_j \leq s_i \wedge e_j > s_i \wedge s_i < e_i) \wedge r_{ij} = r_j) \vee ((s_j > s_i \vee e_j \leq s_i \vee s_i = e_i) \wedge r_{ij} = 0)$

For each task i

$sr_i = r_{i1} + \cdots + r_{in}$

CHECK

$$(s_1 + d_1 = e_1 \wedge \cdots \wedge s_n + d_n = e_n \wedge sr_1 \leq limit \wedge \cdots \wedge sr_n \leq limit) \Leftrightarrow b$$

$$\text{CYCLE}(nc, \langle s_1, \ldots, s_n \rangle)$$

PFD

$$\text{SORT}(S, \langle r_1, \ldots, r_n \rangle)$$

PFD for *alldifferent*

for each s_i

$$\text{ELEMENT}(i, S, s_{i,1}) \wedge \text{ELEMENT}(s_{i,1}, S, s_{i,2}) \wedge \cdots \wedge \text{ELEMENT}(s_{i,n-2}, S, s_{i,n-1})$$

Extracting i-th cycle

$$\text{MINIMUM}(name_i, \langle i, s_{i,1}, s_{i,2}, \ldots, s_{i,n-1} \rangle)$$

Getting a unique representative for i-th cycle

$$\text{NVALUE}(nb, \langle name_1, \ldots, name_n \rangle)$$

Counting the number of distinct representatives

CHECK

$$(r_1 < r_2 \wedge \cdots \wedge r_{n-1} < r_n \wedge$$

Check for *alldifferent*

$$nc = nb) \Leftrightarrow b$$

Check on number of cycles

- Introduction and motivation
- How to derive reified global constraints
- Reification of core global constraints
- **Categories used in reifying global constraints**
- A classification of g.c. wrt. reification
- Conclusion

Categories (automata)

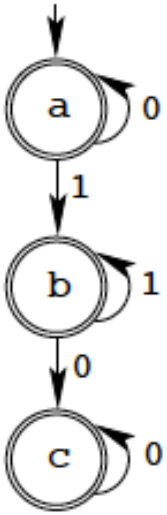
Automata

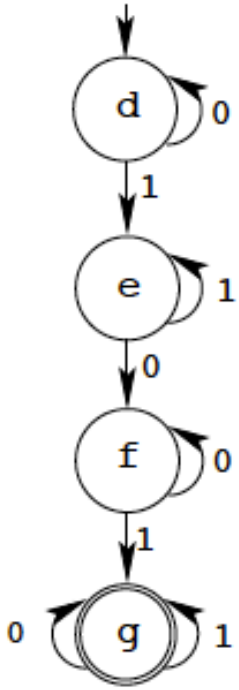
(with, without counters)

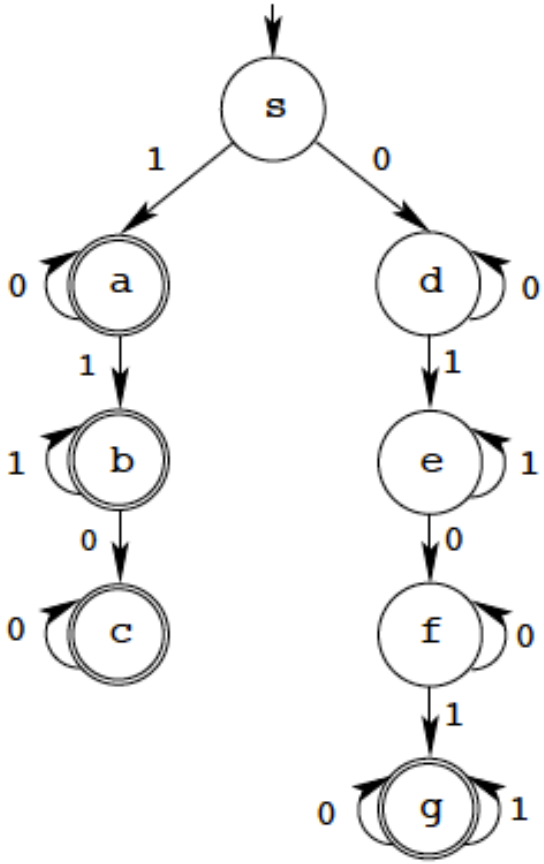
(with, without signature constraint)

can construct reification from the automata

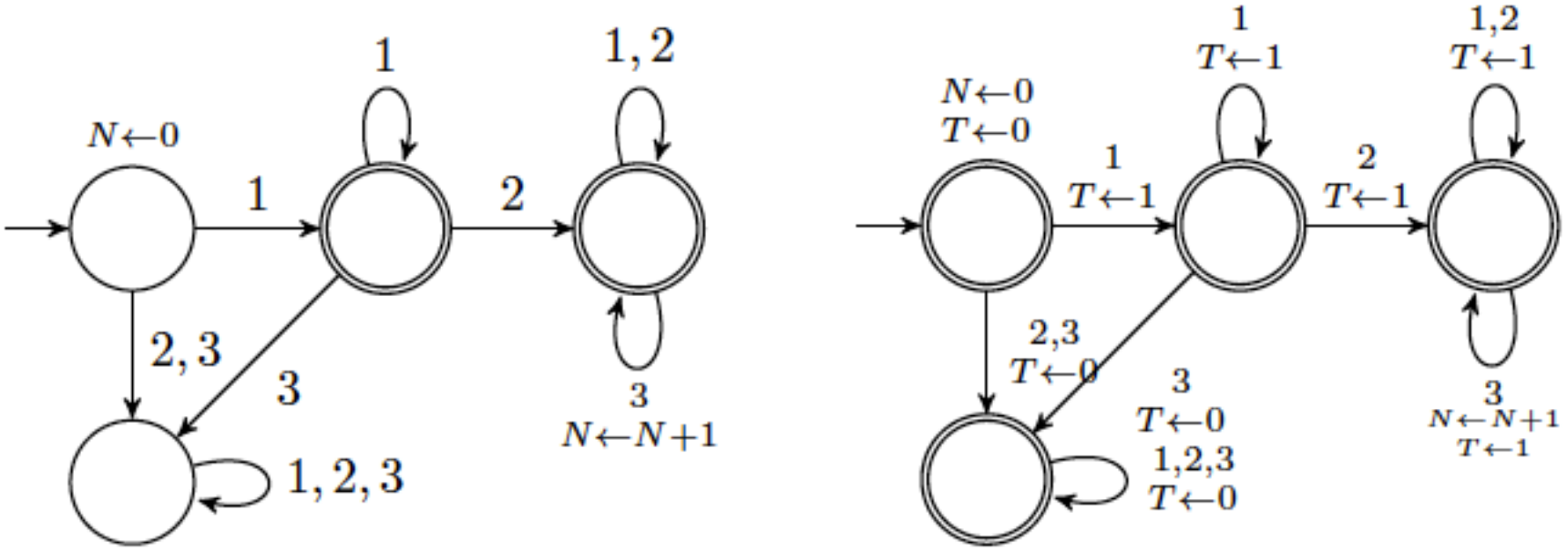
Example (*automaton without counter*)





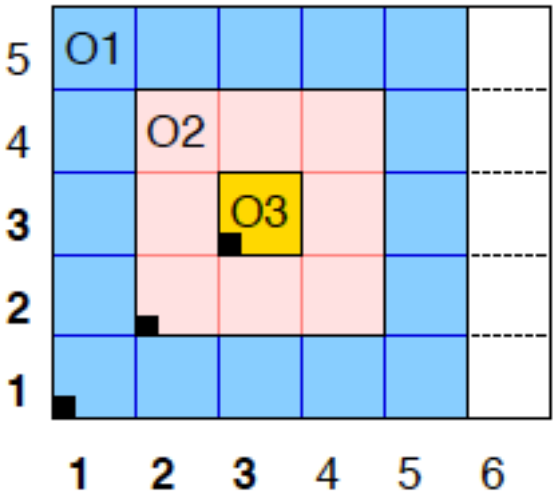


Example (*automaton with counter*)



Category (*Qlogic*)

contain_sboxes(K,DIMS, OBJECTS, SBOXES)



- $\texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D})$
- $\texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D}) + \texttt{S1.l}(\texttt{D})$
- $\texttt{contains\_sboxes}(\texttt{Dims},\texttt{O1},\texttt{S1},\texttt{O2},\texttt{S2}) \stackrel{\text{def}}{=}$

$$\forall \texttt{D} \in \texttt{Dims} \quad \bigwedge \left(\begin{array}{l} \texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) < \\ \texttt{origin}(\texttt{O2},\texttt{S2},\texttt{D}) \\ \texttt{end}(\texttt{O2},\texttt{S2},\texttt{D}) < \\ \texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \end{array} , \right)$$

- $\texttt{contains\_objects}(\texttt{Dims},\texttt{O1},\texttt{O2}) \stackrel{\text{def}}{=}$

$$\forall \texttt{S1} \in \texttt{sboxes}([\texttt{O1.sid}]) \quad \exists \texttt{S2} \in \texttt{sboxes}\left(\left[\texttt{O2.sid} \right] \right) \quad \texttt{contains\_sboxes} \left(\begin{array}{l} \texttt{Dims}, \\ \texttt{O1}, \\ \texttt{S1}, \\ \texttt{O2}, \\ \texttt{S2} \end{array} \right)$$

- $\texttt{all\_contains}(\texttt{Dims},\texttt{OIDS}) \stackrel{\text{def}}{=}$

$$\forall \texttt{O1} \in \texttt{objects}(\texttt{OIDS}) \quad \forall \texttt{O2} \in \texttt{objects}(\texttt{OIDS}) \quad \texttt{O1.oid} < \texttt{O2.oid} \Rightarrow \texttt{contains\_objects} \left(\begin{array}{l} \texttt{Dims}, \\ \texttt{O1}, \\ \texttt{O2} \end{array} \right)$$

- $\texttt{all\_contains}(\texttt{DIMENSIONS},\texttt{OIDS})$

Category (*sort*)

Many constraints on a collection of variables become much simpler to define if you sort the variables

(use the sort constraint for that which is a PFD)

The nice point is that these reformulations are linear in size wrt. number of variables,

e.g., get a linear size reformulation for alldifferent (*unlike the naïve one, the one using gcc where values are made explicit, the one of C. Bessière and al.*)

- Introduction and motivation
- How to derive reified global constraints
- Reification of core global constraints
- Categories used in reifying global constraints
- **A classification of g.c. wrt. reification**
- Conclusion

Table providing reification for 313 out of 381 contraints (82 %)

Global Constraint	Categories	Comment
ABS_VALUE(y, x)	PFD, Logic	$(y = \lvert x \rvert) \Leftrightarrow b$
ALLDIFF_AT_LEAST_K_POS$(k, \langle \langle \mathbf{v} \rangle^m \rangle^n)$	Logic	$\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \sum_{\ell=1}^{m} (v_{i,\ell} \neq v_{j,\ell}) \geq k$
ALL_EQUAL$(\langle \mathbf{v} \rangle^n)$	Logic	$(v_1 = v_2 \wedge \cdots \wedge v_{n-1} = v_n) \Leftrightarrow b$
ALL_INCOMPARABLE$(\langle \langle \mathbf{v} \rangle^m \rangle^n)$	Conj	conjunction of INCOMPARABLE constraints on pairs of vectors
ALL_MIN_DIST$(md, \langle \mathbf{v} \rangle^n)$	Sort	SORT$(\langle \mathbf{v} \rangle^n, \langle \mathbf{s} \rangle^n) \wedge (s_2 - s_1 \geq md \wedge \cdots \wedge s_n - s_{n-1} \geq md) \Leftrightarrow b$
.....	.....	.....
USES$(\langle \mathbf{u} \rangle^m, \langle \mathbf{v} \rangle^n)$	GenPat	Let $\langle \mathbf{w} \rangle^p$ be the values that can be assigned to the variables of $\langle \mathbf{u} \rangle^m$ and $\langle \mathbf{v} \rangle^n$: GCC$(\langle \mathbf{u} \rangle^m, \langle w_i, o_i \rangle_{i=1}^p) \wedge$ GCC$(\langle \mathbf{v} \rangle^n, \langle w_i, q_i \rangle_{i=1}^p) \wedge ((q_1 = 0 \vee o_1 > 0) \wedge \cdots \wedge (q_p = 0 \vee o_p > 0)) \Leftrightarrow b$
VALLEY	Auto(1,2)	
VEC_EQ_TUPLE$(\langle \mathbf{v} \rangle^n, \langle \mathbf{t} \rangle^n)$	Logic	$(v_1 = t_1 \wedge \cdots \wedge v_n = t_n) \Leftrightarrow b$
VISIBLE	?	
WEIGHTED_PARTIAL_ALLDIFF	?	
XOR	PFD, Auto(0,0)	

- Introduction and motivation
- How to derive reified global constraints
- Reification of core global constraints
- Categories used in reifying global constraints
- A classification of g.c. wrt. reification
- **Conclusion**

Conclusion

Exploit the **determine** and **test** scheme for defining global constraints

- A simple way for providing reification (*and negation, and reformulation*)
- Could also be used for measuring cost violation of global constraints (*evaluate cost related to the check part*)

Observations

- Reasonable for a number of categories (*automata, PFD*) where you get things for free (*using the meta data of the catalogue*)

- Given a PFD constraint and its filtering algorithm you should get a similar pruning power for the reified version

Observations

- Huge for graph constraints (*could be lowered*)
  From a practical point of view cubic size reformulations are useless

- Reformulations of C. Bessière *et al.* for *alldifferent*, *global cardinality*, … can be unfolded to make PFD explicit

Conclusion

- You may (perhaps) exploit the constraints network associated with these reifications?

The PFD part looks maybe similar to a dag?

Thanks

Technical report available at

http://soda.swedish-ict.se/5194/