Overall procedure

Combining all of the above mentioned step, the overall projection procedure can thus be described with the following pseudocode:

function PROJECT(S,Y)
  $(S, Y) \leftarrow$ PREPROCESS(S, Y)
  $(S, Y) \leftarrow$ GAUSS-ELIM(S, Y)
  while $Y \neq \emptyset$ **do**
    $(S, Y, New) \leftarrow$ FME-SINGLEVAR(S, Y)
    $S \leftarrow$ REMOVEREDUNDANCY(S, New)
  return S

The sub-algorithms used are:

function PREPROCESS(S, Y)
  do
    $(S, Y) \leftarrow$ remove unused variables (in Y) and and empty (in)equalities
    $(S, Y) \leftarrow$ do FME on variables only occurring with only one sign
    Update bounds for all variables and the left-hand-side of (in)equalities
    $S \leftarrow$ remove syntactically redundant inequalities
  while A variable or (in)equality is removed, a variable is replaced with a value, or a bound is made stricter
  $S \leftarrow$ add inequalities expressing the non-trivial bounds for the variables
  return (S, Y)

function GAUSS-ELIM(S, Y)
  while Y contains a variable used in an equality in S **do**
    $Y' \leftarrow$ the variables in Y that are used in any equality in S
    $x \leftarrow$ the variable in Y' used by the fewest (in)equalities in S
    $e \leftarrow$ the equality in S using x that uses the fewest variables
    Remove e from S and x from Y
    Isolate x in e and substitute in all (in)equalities in S
  return (S, Y)

function FME-SINGLEVAR(S, Y)
  $x \leftarrow$ the variable in Y that minimizes $|Pos_S(x) \cdot Neg_S(x) - Pos_S(x) - Neg_S(x)|$
  $S' \leftarrow \{ \, i_{p,n,x} \mid p \in Pos_S(x), n \in Neg_S(x) \, \}$
  return $(Zero_S(x) \cup S', Y \setminus \{x\}, S')$

function REMOVEREDUNDANCY(S, N)
  Examine each inequalities in N in parallel:
    $R \leftarrow$ the inequalities that are strictly redundant w.r.t. S
    $A \leftarrow$ the inequalities that are almost redundant w.r.t. S
  $S \leftarrow S \setminus R$
  Examine all inequality in A, sequentially:
    $A' \leftarrow$ the inequalities that are almost redundant w.r.t. S
return $S \setminus (R \cup A')$

Nåh ja, og så er der lige list "clean-up" ind imellem. Tror ikke, jeg har brug for at beskrive det med $\kappa - values$, da det ikke er aktuelt i de konkrete projektioner...(?)

Each time we further decompose a system, we use a partitioning of the subsystems $S_1^l, \ldots, S_{k_l}^l$ to create k_{l+1} new subsystems $S_1^{l+1}, \ldots, S_{k_{l+1}}^{l+1}$. This creates a new "level" of subsystems and results in a tree structure of smaller inequality systems, where each node is associated with a subsystem and a set of variables that should be eliminated from the system. The leafs are associated with the systems $S_1^0, \ldots, S_k^0$ and the variable sets $Y \cap VAR(S_1), \ldots, Y \cap VAR(S_k)$, respectively, while the root is associated with the the pair $(S_g^K, \{ \, z_{c,j}^K \mid (c, j) \in S_g \times \{1, \ldots, k_K\} \, \} \cup Y')$ for some K, where $Y' = Y \setminus VAR(S_1) \cup \ldots \cup VAR(S_k)$. For $0 < l \leq K$, a node n associated with the system $S_i^l = \{ \, Def(z_{c,i}^l) \mid c \in S_g \, \}$ is associated with the variable set $\{ \, z_{c,j}^{l-1} \mid (c, j) \in S_g \times \{1, \ldots, k_{l-1}\} \, \} \cap VAR(S_i^l)$.

To project S w.r.t. Y we therefore create the tree T as decribed above and ... recursively.. That is, we call PROJECTNODE(root of T), where

```
function ProjectNode(Node n)
    (S, Y) ← the system and variable set associated with n
    if n is a leaf then
        return FM-ProjectionFramework(S, Y)
    else
        for all children m of n do
            S ← S ∪ ProjectNode(m)
        return FM-ProjectionFramework(S, Y)
```

Parallelization

```
function Manager(T tree structure of S)
    Initialize the count of all nodes to 0 and all projections to ∅
    Create w workers, initially idle
    Initialize a queue Q with all leaves in T
    while the projection of T equals ∅ do
        if Q is non-empty then
            As soon as a worker w is idle
            Remove first node n from Q
            Give w n as input
return the projection of the root of T

function Worker(node n)
    the projection of n ← Project(System S of n, elimination variables of n)
    p ← parent of n
    Increase count of p
    if the count of p equals the number of p's children then
        add p to Q
    return to idle state
```