A EXTRACTION & CLEARING OPERATIONS ON TERMS

In this section, we present the definitions of two operations on terms—extraction and clearing—that are not shown in the main text, together with their meta-properties.

**Extraction.** The definition of the extraction operation $t_1 \downarrow t_2$ follows the same structure as the merge operation shown in the main text. This operation is defined only when $t_1$ and $t_2$ are sorted to the same signature and $t_2$ is a mask.

\[
\begin{align*}
\text{nil} \downarrow t &= \text{nil} \quad (1) \\
\text{t} \downarrow \text{nil} &= \text{nil} \quad (2) \\
(r_1 \mapsto v_1 :: t_1) \downarrow (r_2 \mapsto v_2 :: t_2) &= \begin{cases} 
\text{if } r_1 = r_2 \text{ then } r_1 \mapsto (v_1 \downarrow v_2 :: (t_1 \downarrow t_2)) \\
\text{else if } r_1 < r_2 \text{ then } t_1 \downarrow (r_2 \mapsto v_2 :: t_2) \\
\text{else } (r_1 \mapsto v_1 :: t_1) \downarrow t_2 
\end{cases} \quad (3)
\end{align*}
\]

\[
\begin{align*}
v_1 \downarrow \text{mask} &= v_1 \quad (4) \\
\text{nested}(t_r) \downarrow \text{nested}(m_r) &= \text{nested}(t_r \downarrow m_r) \quad (5) \\
\text{mask} \downarrow \text{nested}(m_r) &= \text{nested}(m_r) \quad (6)
\end{align*}
\]

An auxiliary function $v_1 \downarrow \downarrow v_2$ handles the extraction operation over data values (in 3): If $v_2$ is mask, $v_1$ needs be extracted and thus is returned (4). Otherwise, $v_2$ must be a nested mask $\text{nested}(m_r)$. If $v_1$ is also nested, then the operation applies to the nested terms recursively (5). Otherwise, $v_1$ must be mask. In this case, the algorithm needs to extract the values of the ranges specified in $m_r$, which is just $m_r$ itself (6).

The following theorem states that on well-sorted terms that satisfy the preconditions of $\downarrow$, the operation preserves sorts, and that, *semantically*, $\downarrow$ simulates the bitwise & operation.

**Theorem A.1.** Suppose $\vdash t_1 : \sigma$ and $\vdash t_2 : \sigma$. If $\text{is\_mask}(t_2)$, then:

1. $\vdash t_1 \downarrow t_2 : \sigma$;
2. $[t_1 \downarrow t_2] = [t_1] \& [t_2]$, where $\&$ is the bitwise AND operator on Coq integers.

**Clearing.** The definition of the clearing operation $t_1 \downarrow t_2$ also follows the same structure as the merge operation shown in the main text. Note the duality to extraction since clearing is the opposite—instead of extracting bitfields, it unsets those values. This operation is defined only when $t_1$ and $t_2$ are sorted to the same signature and $t_2$ is a mask.

Authors’ addresses: Fengmin Zhu, MPI-SWS, Saarland Informatics Campus, Germany, paulzhu@mpi-sws.org; Michael Sammler, MPI-SWS, Saarland Informatics Campus, Germany, msammler@mpi-sws.org; Rodolphe Lepigre, MPI-SWS, Saarland Informatics Campus, Germany, lepigre@mpi-sws.org; Derek Dreyer, MPI-SWS, Saarland Informatics Campus, Germany, dreyer@mpi-sws.org; Deepak Garg, MPI-SWS, Saarland Informatics Campus, Germany, dg@mpi-sws.org.
An auxiliary function \( v_1 \downarrow \sim v_2 \) handles the clearing operation over data values (in 9): If \( v_2 \) is mask, \( v_1 \) needs be cleared (or unset) and thus the algorithm returns \( \text{data}(0) \) (10). If both are nested terms, then the operation applies to the nested terms recursively (11). Finally, if \( v_1 \) is mask and \( v_2 \) is a nested mask \( m_r \), then the algorithm needs to clear all the ranges mentioned by \( m_r \) in \( v_1 \). Since \( v_1 \) is a mask, this means the result will be the negation of \( m_r \) (denoted by \( \text{comp}(m_r) \)) wrapped by \( \text{nested} \) (12).

The following theorem states that on well-sorted terms that satisfy the preconditions of \( \downarrow \sim \), the operation preserves sorts, and that, semantically, \( \downarrow \sim \) simulates the bitwise \&~ operation.

**Theorem A.2.** Suppose \( \vdash t_1 : \sigma \) and \( \vdash t_2 : \sigma \). If \( \text{is\_mask}(t_2) \), then:

1. \( \vdash t_1 \downarrow \sim t_2 : \sigma \);
2. \([t_1 \downarrow \sim t_2] = [t_1] \& \sim [t_2] \), where \( \sim \) is the bitwise NOT operator on Coq integers.