Formal specification and refinement of a Heap specification (V1)

Leo Freitas, Cliff Jones

October 8, 2013

Contents

Chapter 1

Some Dependencies

$\textbf{section}\ \mathit{arithmetic}\ \textbf{parents}\ \mathit{standard\_toolkit}$

This specification describes ...

1.1 Arithmetic

These theorems are simple arithmetic transformations that are often useful when reasoning about non-linear equations in formulae. They are all trivial consequences of integer (Peano's) arithmetic.

Initially, we had the theorems laid out with quantified $\mathbb{Z}$ variables. This is not very helpful as it leads to type-checking proof obligations over the pattern matched expressions to i and j. So, instead of

$$\forall i, j : \mathbb{Z} \mid i \geq j \bullet \neg\, i < j$$

we prefer to say simply

$$i \geq j \Rightarrow \neg\, i < j$$

in order to take advantage of the "joker" (place-holder) implicitly (universally) quantified variables, which can only be typed as $\mathbb{A}$ (or $\mathbb{Z}$ in Z/Eves). That is, the former version would lead to a proof obligation that whatever i and j pattern matches to, we would need to show that $i, j \in \mathbb{Z}$, which can be as complex as the expressions for i and j. Using the "joker"-pattern-matching, we avoid this need altogether. It is possible to do it for arithmetic, since it is embedded within the prover. For other situations, one could only hope for weaker type-checking conditions, rather than avoid it altogether, like we are doing here.

[Negate $<$ to $\geq$]

theorem disabled rule lLessNeg
$i \geq j \Rightarrow \neg\, i < j$

[Negate $>$ to $\leq$]

theorem disabled rule lGreaterNeg
$i \leq j \Rightarrow \neg\, i > j$

[Negate $\leq$ to $>$]

theorem disabled rule lLeqNeg
$i > j \Rightarrow \neg\, i \leq j$

[Negate $\geq$ to $<$]

theorem disabled rule lGeqNeg
$i < j \Rightarrow \neg\, i \geq j$

[Flip $<$ to $>$]

theorem rule lLessFlip
$j > i \Rightarrow i < j$

[Flip $>$ to $<$]

theorem disabled rule lGreaterFlip
$j < i \Rightarrow i > j$

[Flip $\leq$ to $\geq$]

theorem disabled rule lLeqFlip
$j \geq i \Rightarrow i \leq j$

[Flip $\geq$ to $\leq$]

theorem disabled rule lGeqFlip
$j \leq i \Rightarrow i \geq j$

For arithmetic "promotion", because we apply it to an operator $(\_ + \_)$-plus that expects $\mathbb{A}$ ($\mathbb{Z}$ in Z/Eves), we do need to add the types for i and j. Otherwise, the decision procedures for arithmetic cannot decide weather to treat i and j as numbers or not.

[Promote $<$ into $\leq$]

theorem disabled rule lLessPromote
$\forall i, j : \mathbb{Z} \mid 1 + i \leq j \bullet i < j$

[Promote $>$ into $\geq$]

theorem disabled rule lGreaterPromote
$\forall i, j : \mathbb{Z} \mid i \geq 1 + j \bullet i > j$

1.2 Arithmetic Proofs

proof[*lLessNeg*]
simplify;
∎

proof[*lGreaterNeg*]
simplify;
∎

proof[*lLeqNeg*]
simplify;
∎

proof[*lGeqNeg*]
simplify;
∎

proof[*lLessFlip*]
simplify;
∎

proof[*lGreaterFlip*]
simplify;
∎

proof[*lLeqFlip*]
simplify;
∎

proof[*lGeqFlip*]
simplify;
∎

proof[*lLessPromote*]
simplify;
∎

proof[*lGreaterPromote*]
simplify;
∎

section *sets* **parents** *standard_toolkit*

1.3 Big cup

$[XX]$

$bigU : \mathbb{P}(\mathbb{P}\ XX) \rightarrow \mathbb{P}\ XX$

$\langle\!\langle$ disabled rule dBigU $\rangle\!\rangle$
$\forall SS : \mathbb{P}(\mathbb{P}\ XX) \bullet bigU\ SS = \{v : XX \mid \exists S : \mathbb{P}\ XX \mid S \in SS \bullet v \in S\}$

theorem rule dlBigCupAsBigU $[XX]$
$\forall SS : \mathbb{P}(\mathbb{P}\ XX) \bullet \bigcup SS = bigU\ SS$

An easy lemma to have BigU just like $\bigcup$

theorem disabled rule dlInBigU $[XX]$
$\forall SS : \mathbb{P}(\mathbb{P}\ XX) \bullet x \in bigU\ SS \Leftrightarrow (\exists ss : SS \bullet x \in ss)$

theorem disabled rule dlInPowerBigU $[XX]$
$\forall SS : \mathbb{P}(\mathbb{P}\ XX) \mid x \in SS \bullet x \in \mathbb{P}(bigU\ SS)$

1.4 Ranges

theorem disabled rule dlRangeCapLeft
$\forall A, B, C, D : \mathbb{Z} \mid B < C \bullet (A \mathinner{..} B) \cap (C \mathinner{..} D) = (C \mathinner{..} B)$

theorem disabled rule dlRangeCapRight
$\forall A, B, C, D : \mathbb{Z} \mid D < A \bullet (A \mathinner{..} B) \cap (C \mathinner{..} D) = (A \mathinner{..} D)$

theorem rule dlRangeCapEmpty
$\forall A, B, C, D : \mathbb{Z} \mid B < A \lor D < C \lor B < C \lor D < A \bullet (A \mathinner{..} B) \cap (C \mathinner{..} D) = \{\}$

theorem disabled rule dlRangeSumSubset
$\forall a, b, x, y : \mathbb{N} \mid x \leq a \land a + b \leq x + y \bullet a \mathinner{..} a + b - 1 \subseteq x \mathinner{..} x + y - 1$

theorem disabled rule dlRangeDifference
$\forall A, B, C : \mathbb{Z} \mid A < B \bullet (1 + B \mathinner{..} C) = (A \mathinner{..} C) \setminus (A \mathinner{..} B)$

1.5 Proofs

1.5.1 Bigcup proofs

proof[$dlBigCupAsBigU$]
apply extensionality;
prove;
apply dBigU to expression $bigU[XX]\ SS$;
prove;
cases;
apply inBigcup to predicate $x \in \bigcup[XX]\ SS$;
prove;
split $x \in XX$;
rewrite;
cases;
instantiate $S == B$;
prove;
next;
rearrange;
split $\exists\ S\_\_0 : \mathbb{P}\ XX \bullet S\_\_0 \in SS \land x \in S\_\_0$;
simplify;
prove;
next;
instantiate $B == S$;
prove;
next;
∎

proof[$dlInBigU$]
split $x \in bigU[XX]\ SS$;
cases;
rewrite;
apply $dBigU$;
prove;
instantiate $ss == S$;
rewrite;
next;
rewrite;
rearrange;
split $(\exists\ ss : SS \bullet x \in ss)$;
rewrite;
apply $dBigU$;
prove;
instantiate $S == ss$;
prove;
next;
∎

Andrius: how does CZT parses this first proof command?

proof[$dlInPowerBigU$]
apply inPower to predicate $x \in \mathbb{P}\ (bigU\ [XX]\ SS)$;
apply $dBigU$;
prove;
instantiate $S == x$;
prove;
∎

1.5.2 Range proofs

proof[*dlRangeCapLeft*]
apply extensionality;
prove;
■

proof[*dlRangeCapRight*]
apply extensionality;
prove;
■

proof[*dlRangeCapEmpty*]
split $B < A$;
prove;
split $D < C$;
prove;
apply lLessNeg to predicate $D < C$;
apply lLessNeg to predicate $B < A$;
simplify;
apply lGeqFlip;
simplify;
split $B < C$;
cases;
apply dlRangeCapLeft;
simplify;
apply rangeNull;
simplify;
next;
apply dlRangeCapRight;
simplify;
apply rangeNull;
simplify;
next;
■

proof[*dlRangeSumSubset*]
prove;
■

proof[*dlRangeDifference*]
apply extensionality;
prove;
■

Chapter 2

Abstract spec — set of *Loc*

2.1 Heap 0 spec

section $HeapCBJ0$ **parents** $arithmetic$

theorem Loc_ vc_ fsb_ horiz_ def
$\exists Loc : \mathbb{P}\,\mathbb{N} \mid true \bullet true$

$Loc == \mathbb{N}$

theorem Free0_ vc_ fsb_ horiz_ def
$\exists Free0 : \mathbb{P}\,\mathbb{P}\,Loc \mid true \bullet true$

$Free0 == \mathbb{P}\ Loc$
$Piece \mathrel{\widehat=} [\,LOC : Loc;\ SIZE : \mathbb{N}\,]$

$locs\_of : Piece \rightarrow \mathbb{P}\ Loc$

$\langle\!\langle$ disabled rule dlLocsOfDef $\rangle\!\rangle$
$\forall p : Piece \bullet locs\_of\ \ p = \{l : Loc \mid \exists i : 0 \mathrel{..} p.SIZE - 1 \bullet i + p.LOC \leq l\}$

theorem grule gLocsOfRelType
$locs\_of \in \lblot LOC : \mathbb{Z};\ SIZE : \mathbb{Z} \rblot \leftrightarrow \mathbb{P}\ \mathbb{Z}$

theorem rule lLocsOfIsTotal
$\forall p : Piece \bullet p \in \operatorname{dom}\ locs\_of$

Heap0
Z_Abs_St
$free0 : Free0$

$true$

$$\begin{array}{|l}
\multicolumn{1}{l}{NEW0} \\ \hline
\Delta Heap0 \\
req? : \mathbb{N} \\
res! : Piece \\ \hline
req? = res!.SIZE \\
free0' = free0 \setminus locs_of\ (res!) \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{DISPOSE0} \\ \hline
\Delta Heap0 \\
ret? : Piece \\ \hline
locs_of\ (ret?) \cap free0 = \varnothing \\
free0' = free0 \cup locs_of\ (ret?) \\ \hline
\end{array}$$

2.2 Lemmas

theorem grule gLocMaxType
$Loc \in \mathbb{P}\ \mathbb{Z}$

theorem grule gLocType
$Loc \in \mathbb{P}\ \mathbb{N}$

theorem frule fPieceLOCMaxType
$p \in Piece \Rightarrow p.LOC \in \mathbb{Z}$

theorem frule fPieceSIZEMaxType
$p \in Piece \Rightarrow p.SIZE \in \mathbb{Z}$

theorem grule gPieceMaxType
$Piece \in \mathbb{P}\,(\langle\!| LOC : \mathbb{Z};\ SIZE : \mathbb{Z} |\!\rangle)$

theorem rule lLocsOfResMaxType
$\forall\, p : Piece \bullet locs\_of\ (p) \in \mathbb{P}\ \mathbb{Z}$

```
begin{theorem}{dlLocsOfPropInductLHS}
% \forall LOC, SIZE: \nat @ \nat_1 \subseteq \{ k: \nat_1 | \\
% \t1 \forall i: \nat | i+LOC < k \land SIZE > 0 \implies i \geq 1  \\
% \t2 @ i+LOC \upto k \subseteq LOC\upto(LOC+SIZE-1) \} %k < i+LOC \lor k \leq LOC+SIZE-1 \}
\forall LOC, SIZE, i: \nat | i < SIZE @ \nat \subseteq \{ k: \nat | i+LOC \leq k \land  k+1 \leq LOC+SIZE \}
end{theorem}k < i + LOC \lor
```

theorem disabled rule dlLocsOfProp
$\forall\, p : Piece \bullet locs\_of\ (p) = p.LOC \mathrel{..} p.LOC + p.SIZE - 1$

2.3 VCs

$$\begin{array}{|l} \multicolumn{1}{l}{PieceFSBSig} \\ \hline \quad Piece \\ \hline Piece \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{Heap0FSBSig} \\ \hline \quad Heap0 \\ \hline Heap0 \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{NEW0FSBSig} \\ \hline \quad Heap0 \\ req? : \mathbb{N} \\ \hline \exists\, t : Piece \bullet t.SIZE = req? \land locs\_of\ (t) \subseteq free0 \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{DISPOSE0FSBSig} \\ \hline \quad Heap0 \\ ret? : Piece \\ \hline locs\_of\ (ret?) \cap free0 = \varnothing \\ \hline \end{array}$$

theorem Piece_ vc_ fsb_ state
$\exists\, PieceFSBSig \mid true \bullet true$

theorem Heap0_ vc_ fsb_ state
$\exists\, Heap0FSBSig \mid true \bullet true$

theorem NEW0_ vc_ fsb_ pre
$\forall\, NEW0FSBSig \mid true \bullet \mathbf{pre}\ NEW0$

theorem DISPOSE0_ vc_ fsb_ pre
$\forall\, DISPOSE0FSBSig \mid true \bullet \mathbf{pre}\ DISPOSE0$

2.4 Proofs

proof[*locsOf*\$*domainCheck*]
with enabled (Loc) prove by reduce;
■

proof[*NEW0*\$*domainCheck*]
with enabled (Loc) prove by reduce;
■

proof[$DISPOSE0\$domainCheck$]
with enabled (Loc) *prove by reduce*;
∎

proof[$NEW0FSBSig\$domainCheck$]
with enabled (Loc) *prove by reduce*;
∎

proof[$DISPOSE0FSBSig\$domainCheck$]
with enabled (Loc) *prove by reduce*;
∎

proof[$Loc\_vc\_fsb\_horiz\_def$]
instantiate $Loc == \{0\}$;
prove;
∎

proof[$Free0\_vc\_fsb\_horiz\_def$]
instantiate $Free0 == \varnothing$;
prove;
∎

proof[$Piece\_vc\_fsb\_state$]
instantiate $LOC == 0, SIZE == 0$;
with enabled (Loc) *prove by reduce*;
∎

proof[$Heap0\_vc\_fsb\_state$]
instantiate $free0 == \varnothing$;
prove by reduce;
∎

proof[$NEW0\_vc\_fsb\_pre$]
prove by reduce;
instantiate $res! == t$;
prove;
∎

proof[$DISPOSE0\_vc\_fsb\_pre$]
prove by reduce;
∎

2.4.1 Lemmas proofs

proof[$gLocMaxType$]
$with\ enabled\ (Loc)\ prove\ by\ reduce;$
$\blacksquare$

proof[$gLocType$]
$with\ enabled\ (Loc)\ prove\ by\ reduce;$
$\blacksquare$

proof[$fPieceLOCMaxType$]
$with\ enabled\ (Piece\$member)\ prove\ by\ reduce;$
$\blacksquare$

proof[$fPieceSIZEMaxType$]
$with\ enabled\ (Piece\$member)\ prove\ by\ reduce;$
$\blacksquare$

proof[$gPieceMaxType$]
$prove;$
$\blacksquare$

proof[$lLocsOfResMaxType$]
$apply\ dlLocsOfDef;$
$prove;$
$\blacksquare$

proof[$gLocsOfRelType$]
$use\ locsOf\$declaration;$
$invoke\ (\_ \rightarrow \_);$
$invoke\ (\_ \nrightarrow \_);$
$invoke\ (\_ \leftrightarrow \_);$
$rewrite;$
$trivial\ rewrite;$
$prenex;$
$apply\ inPower;$
$prenex;$
$instantiate\ e\_\_0\ ==\ e;$
$apply\ inCross2;$
$with\ enabled\ (Loc)\ prove\ by\ reduce;$
$\blacksquare$

THIS IS RIDICULOUS! THERE IS SOME MISSING TYPE BRIDGE

proof[$lLocsOfIsTotal$]
$use\ locsOf\$declaration;$
$invoke\ (\_ \rightarrow \_);$
$apply\ inDom;$
$rewrite;$
$instantiate\ x\ ==\ p;$
$prove;$
$instantiate\ y\_\_1\ ==\ y;$
$with\ enabled\ (Loc)\ prove\ by\ reduce;$
$\blacksquare$

```
begin{zproof}[dlLocsOfPropInductLHS]
apply natInduction;
conjunctive;
cases;
% type
rewrite;
next;
% base case
prove;
next;
% inductive case
prove;
split x = LOC+SIZE-1;
rewrite;
end{zproof}   to predicate \nat \subseteq \{k: \nat | i + LOC \upto k \subseteq LOC \upto (LOC + S
```

This proof is incomplete

proof[*dlLocsOfProp*]

apply extensionality;
apply dlLocsOfDef;
with enabled (*Loc*, *Piece$member*) *with disabled* (*inRange*) *prove by reduce*;
split $LOC = 0 \lor SIZE = 0$;
cases;
with disabled (*inRange*) *prove*;
split $LOC = 0$;
with disabled (*inRange*) *prove*;
cases;

next;
next;
split $LOC > 0 \land SIZE > 0$;
cases;
next;
prove;
next;
rewrite;
apply extensionality;
apply dlLocsOfDef;
split $SIZE = 0$;
with disabled (*inRange*) *rewrite*;
cases;
prove;
apply inPower;
rewrite;
instantiate $e == x$;
rewrite;
instantiate $i\_\_0 == i$;
rewrite;
next;
cases;
rewrite;
next;
rewrite;
instantiate $i\_\_0 == SIZE - 1$;
prove;
split $y = -1 + (LOC + SIZE)$;
prove;
■

ANNOYING INDUCTION NEEDED? MAYBE JUST TAKE IT AS TRUE OR DEFINED AS

Declarations	**This Chapter**	**Globally**
Unboxed items	4	6
Axiomatic definitions	1	1
Generic axiomatic defs.	0	1
Schemas	7	7
Generic schemas	0	0
Theorems	15	33
Proofs	20	38
Total	**47**	**86**

Table 2.1: Summary of Z declarations for Chapter 2.

Chapter 3

Intermediate design — set of *Piece*

3.1 Heap CBJ version 1

section $HeapCBJ1$ **parents** $HeapCBJ0, sets$

relation$(invFree1\ \_)$

$$
\begin{aligned}
&invFree1\ \_ == \{fr : \mathbb{P}\ Piece \mid \forall p1, p2 : fr \bullet \\
&\quad p1 = p2 \lor \\
&\quad (locs\_of\ (p1) \cap locs\_of\ (p2) = \varnothing \land \\
&\quad p1.LOC + p1.SIZE \neq p2.LOC)\} \\
&Free1 == \{ps : \mathbb{P}\ Piece \mid ps \in \mathbb{F}\ Piece \land \\
&invFree1\ (ps)\}
\end{aligned}
$$

theorem frule fFree1ElemMaxType
$f \in Free1 \Rightarrow f \in \mathbb{P}\ \langle\!|LOC : \mathbb{Z};\ SIZE : \mathbb{Z}|\!\rangle$

theorem frule fFree1ElemType
$f \in Free1 \Rightarrow f \in \mathbb{P}\ Piece$

$locs : \mathbb{P}\ Piece \rightarrow \mathbb{P}\ Loc$

$\langle\!\langle$ disabled rule dlLocsDef $\rangle\!\rangle$
$\forall f : \mathbb{P}\ Piece \bullet locs\ f = \bigcup \{p : Piece \mid p \in f \bullet locs\_of\ (p)\}$

theorem grule gLocsRelType
$locs \in \mathbb{P}\ (\langle\!|LOC : \mathbb{Z};\ SIZE : \mathbb{Z}|\!\rangle) \leftrightarrow \mathbb{P}\ \mathbb{Z}$

theorem rule lLocsIsTotal
$\forall f : \mathbb{P}\ Piece \bullet f \in \mathrm{dom}\ locs$

Heap1

$free1 : Free1$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NEW1}} \\
\Delta Heap1 \\
req? : \mathbb{N} \\
res! : Piece \\
\hline
res!.SIZE = req? \\
\exists\, p, rem : Piece \bullet p \in free1 \land \\
locs\_of\,(res!) \subseteq locs\_of\,(p) \land \\
locs\_of\,(rem) = locs\_of\,(p) \setminus locs\_of(res!) \land \\
free1' = (free1 \setminus \{p\}) \cup \{rem\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{DISPOSE1}} \\
\Delta Heap1 \\
ret? : Piece \\
\hline
locs\_of\,(ret?) \cap locs\,(free1) = \varnothing \\
locs\,(free1') = locs\,(free1) \cup locs\_of(ret?) \\
\hline
\end{array}
$$

3.2 Lemmas

theorem disabled rule dlInFree1

$$f \in Free1 \Leftrightarrow (f \in \mathbb{P}\ Piece \land f \in \mathbb{F}\ Piece \land invFree1\,(f))$$

theorem disabled rule dlInInvFree1

$$
\begin{array}{l}
invFree1\,(fr) \Leftrightarrow (fr \in \mathbb{P}\ Piece \land \forall\, p1, p2 : fr \mid \\
\quad \neg\ p1 = p2 \bullet \\
\quad (locs\_of\,(p1) \cap locs\_of\,(p2) = \{\} \land \\
\quad p1.LOC + p1.SIZE \neq p2.LOC))
\end{array}
$$

theorem rule dlLocsOfCapEmpty

$$
\begin{array}{l}
\forall\, p, q : Piece \mid p.SIZE = 0 \lor q.SIZE = 0 \lor p.LOC + p.SIZE \leq q.LOC \\
\quad \lor\ q.LOC + q.SIZE \leq p.LOC \bullet \\
\qquad locs\_of\,(p) \cap locs\_of\,(q) = \{\}
\end{array}
$$

theorem rule lFree1UnitUnionInType

$$\forall\, f : Free1;\ p : Piece \bullet \{p\} \cup f \in Free1$$

theorem rule lFree1UnitReductionInType

$$\forall\, f : Free1;\ p : Piece \bullet f \setminus \{p\} \in Free1$$

These lemmas are useful to avoid expanding *Piece*

theorem frule fPieceLOCProp

$$p \in Piece \Rightarrow p.LOC \geq 0$$

theorem frule fPieceSIZEProp

$$p \in Piece \Rightarrow p.SIZE \geq 0$$

theorem disabled rule dlLocsOfWithin

$$
\begin{array}{l}
\forall\, p, q : Piece \mid p.LOC \leq q.LOC \land \\
\quad q.LOC + q.SIZE \leq p.LOC + p.SIZE \bullet \\
\qquad locs\_of\,(q) \in \mathbb{P}\,(locs\_of\,(p))
\end{array}
$$

theorem disabled rule dlLocsEmptyRemainder
$\forall\, rem : Piece \mid rem.SIZE = 0 \bullet locs\_of\ \ rem = \varnothing$

theorem disabled rule dlRangeDifferenceV1
$\forall\, qLOC, qSIZE, rSIZE : \mathbb{N} \mid rSIZE \leq qSIZE \land 0 < rSIZE \bullet$
$(1 + qLOC + rSIZE - 1 \mathrel{..} qLOC + qSIZE - 1) =$
$(qLOC \mathrel{..} qLOC + qSIZE - 1) \setminus (qLOC \mathrel{..} qLOC + rSIZE - 1)$

qLOC+rSIZE-1+1 .. qLOC+qSIZE-1 = qLOC .. qLOC+qSIZE-1 qLOC .. qLOC+rSIZE-1

theorem disabled rule dlLocsRemainder
$\forall\, rem, res, q : Piece \mid rem.LOC = q.LOC + res.SIZE \land$
$rem.SIZE = q.SIZE - res.SIZE \land$
$q.SIZE \geq res.SIZE \land$
$res.LOC = q.LOC \bullet$
$locs\_of\ rem = locs\_of\ q \setminus locs\_of\ res$

Was useful in V1; perhaps could be here for the next lemma?

theorem rule lLocsDistUnitDiff
$\forall\, f : Free1 \bullet \forall\, p : f \bullet locs\,(f \setminus \{p\}) = locs\,(f) \setminus locs\_of\,(p)$

theorem rule lLocsDistUnitCup
$\forall\, f : Free1;\ p : Piece \mid locs\_of\ \ p \cap locs\, f = \{\} \bullet locs(\{p\} \cup f) = locs\_of\ p \cup locs\, f$

3.3 VCs

Heap1FSBSig

$Heap1$

$Heap1$

This precondition is open (and sufficient), yet the concrete one to choose for *res* and *rem* are quite prescribed (I think): they need at least to start at the same place (i.e. $res.LOC = p.LOC$), otherwise one would get two pieces remaining rather than one, which is not the case in the postcondition. So this is kind of implicit in the precondition.

NEW1FSBSig

$Heap1$
$req? : \mathbb{N}$

$\exists\, q : Piece \bullet q \in free1 \land q.SIZE \geq req?$

DISPOSE1FSBSig

$Heap1$
$ret? : Piece$

$locs\_of\,(ret?) \cap locs\,(free1) = \varnothing$

theorem Heap1_ vc_ fsb_ state
$\exists\, Heap1FSBSig \mid true \bullet true$

theorem NEW1_ vc_ fsb_ pre
$\forall NEW1FSBSig \mid true \bullet \mathbf{pre}\ NEW1$

theorem DISPOSE1_ vc_ fsb_ pre
$\forall DISPOSE1FSBSig \mid true \bullet \mathbf{pre}\ DISPOSE1$

3.4 Proofs

proof[*invFree$domainCheck*]
with enabled (Loc) prove by reduce;
■

proof[*locs$domainCheck*]
with enabled (Loc) prove by reduce;
■

proof[*NEW1$domainCheck*]
with enabled (Loc) prove by reduce;
■

proof[*DISPOSE1$domainCheck*]
with enabled (Loc) prove by reduce;
■

proof[*DISPOSE1FSBSig$domainCheck*]
with enabled (Loc) prove by reduce;
■

proof[*Heap1_ vc_ fsb_ state*]
instantiate free1 $==$ $\varnothing$;
with enabled (Free1, invFree1 _) prove by reduce;
■

proof[$NEW1\_vc\_fsb\_pre$]

prove by reduce;
split $\neg\ \exists\ result : Piece\ \bullet$
$result = \theta\ Piece[LOC := q.LOC,\ SIZE := req?]$;
cases;
with enabled $(Loc,\ Piece\$member)$ *prove by reduce*;
next;
prenex;
rearrange;
split $q.SIZE = req?$;
rewrite;
cases;
rearrange;
split $\neg\ \exists\ someLoc : Loc;\ remainder : Piece\ \bullet\ remainder = \theta\ Piece[LOC := someLoc,\ SIZE := 0]$;
cases;
with enabled $(Loc,\ Piece\$member)$ *prove by reduce*;
instantiate $someLoc == 0$;
prove;
next;
prenex;
rearrange;
instantiate $res! == result,\ p == q,\ rem == remainder$;
reduce;
apply $dlLocsOfWithin$;
apply $dlLocsEmptyRemainder$ *to expression* $locs\_of\ \ remainder$;
rearrange;
rewrite;
split $\neg\ q = result$;
cases;
with enabled $(Piece\$member)$ *prove by reduce*;
next;
equality substitute $result$;
apply $diffSuperset$;
rewrite;
next;
rearrange;
split $\neg\ \exists\ remainder : Piece\ \bullet\ remainder = \theta\ Piece[LOC := q.LOC + req?,\ SIZE := q.SIZE\ -$
cases;
with enabled $(Loc,\ Piece\$member)$ *prove by reduce*;
next;
prenex;
rearrange;
instantiate $res! == result,\ p == q,\ rem == remainder$;
reduce;
apply $dlLocsOfWithin$;
rewrite;
use $dlLocsRemainder[rem := remainder,\ q := q,\ res := result]$;
rearrange;
rewrite;
next;
■

proof[$DISPOSE1\_vc\_fsb\_pre$]
  prove by reduce;
  instantiate $free1' == free1 \cup \{ret?\}$;
  prove;
  apply cupCommutes to expression $free1 \cup \{ret? \}$;
  with disabled (cupCommutes) rewrite;
  ■

3.4.1 Lemmas proofs

proof[*fFree1ElemMaxType*]
  invoke Free1;
  prove;
  ■

proof[*fFree1ElemType*]
  invoke Free1;
  prove;
  ■

proof[*gLocsRelType*]
  prove;
  ■

proof[*lLocsIsTotal*]
  prove;
  ■

proof[*dlInFree1*]
  invoke Free1;
  prove;
  ■

proof[*dlInInvFree1*]
  invoke (invFree1 _);
  prove;
  ■

proof[*dlLocsOfCapEmpty*]
  apply dlLocsOfProp;
  rewrite;
  apply dlRangeCapEmpty;
  rewrite;
  with normalization prove;
  ■

Needs extra conditions from lFree1UnitUnionInType side conditions

```
begin{theorem}{ulLocsOfDisjFreeInvProp1}
\forall f: Free1; p1, p2: Piece |  p1 \in f \land \lnot p2 \in f @ \\
\t1 \locsOf~p1 \cap \locsOf~p2 = \{\}
end{theorem}
begin{theorem}{ulLocsOfDisjFreeInvProp2}
\forall f: Free1; p1, p2: Piece | p1 \in f \land \lnot p2 \in f @ \\
\t1 \lnot p1 . LOC + p1 . SIZE = p2 . LOC
\end{theorem}
begin{theorem}{ulLocsOfDisjFreeInvProp3}
\forall f: Free1; p1, p2: Piece | p1 \in f \land \lnot p2 \in f @ \\
\t1 \lnot p2 . LOC + p2 . SIZE = p1 . LOC
end{theorem}
```

This proof is incomplete

proof[*lFree1UnitUnionInType*]
prove;
apply dlInFree1;
prove;
apply dlInInvFree1 ;
with disabled (*inCup*) *prove*;
instantiate $p1\_\_0 \;==\; p1,\; p2\_\_0 \;==\; p2$;
with normalization rewrite;
cases;
next;
next;
■

proof[*lFree1UnitReductionInType*]
apply dlInFree1;
prove;
apply dlInInvFree1;
prove;
instantiate $p1\_\_0 \;==\; p1,\; p2\_\_0 \;==\; p2$;
prove;
■

proof[*dlLocsOfWithin*]
apply dlLocsOfProp;
with enabled (*dlRangeSumSubset*) *prove*;
■

proof[*fPieceSIZEProp*]
with enabled (*Piece$member*) *prove by reduce*;
■

proof[*fPieceLOCProp*]
with enabled (*Piece$member*, *Loc*) *prove by reduce*;
■

proof[*dlLocsEmptyRemainder*]
apply dlLocsOfProp;
prove;
■

proof[*dlRangeDifferenceV1*]
apply extensionality;
prove;
■

proof[*dlLocsRemainder*]
apply dlLocsOfProp;
equality substitute res.LOC;
with enabled (*Piece\$member*, *Loc*) *with disabled* (*inRange*) *prove*;
use dlRangeDifferenceV1[*qLOC* := *LOC*__1, *rSIZE* := *SIZE*__0, *qSIZE* := *SIZE*__1];
with enabled (*Loc*) *prove by reduce*;
split $0 < SIZE\_\_0$;
simplify;
apply lLessPromote to predicate $0 < SIZE\_\_0$;
rearrange;
rewrite;
■

proof[*lLocsDistUnitDiff*]
apply dlLocsDef ;
with disabled (*dlBigCupAsBigU*) *prove*;
apply extensionality;
with disabled (*dlBigCupAsBigU*) *prove*;
apply inBigcup;
prove;
cases;
split $\neg\ x \in \mathbb{Z}$;
cases;
prove;
next;
rewrite;
instantiate $B\_\_0 == B$;
prove;
instantiate $p\_\_1 == p\_\_0$;
prove;
apply dlInFree1;
apply dlInInvFree1;
instantiate $p1 == p,\ p2 == p\_\_0$;
prove;
apply extensionality to predicate $locs\_of\ p \cap locs\_of\ p\_\_0 = \{\}$;
prove;
instantiate $x\_\_0 == x$;
prove;
next;
split $\neg\ y \in \mathbb{Z}$;
cases;
prove;
next;
rewrite;
instantiate $B == B\ \_\_0$;
prove;
instantiate $p\_\_1 == p\_\_0$;
prove;
next;
■

proof[*lLocsDistUnitCup*]
apply dlLocsDef;
rewrite;
apply extensionality;
prove;
apply dlInBigU;
prove;
cases;
instantiate $ss\_\_1 == locs\_of\ p\_\_0$;
rewrite;
instantiate $p\_\_1 == p\_\_0$;
rewrite;
next;
split $y \in locs\_of\ p$;
rewrite;
cases;
instantiate $ss == locs\_of\ p$;
rewrite;
instantiate $p\_\_0 == p$;
rewrite;
next;
instantiate $ss\_\_1 == locs\_of\ p\_\_1$;
rewrite;
instantiate $p\_\_1 == p\_\_0$;
rewrite;
next;
$\blacksquare$

Declarations	**This Chapter**	**Globally**
Unboxed items	8	14
Axiomatic definitions	1	2
Generic axiomatic defs.	0	1
Schemas	6	13
Generic schemas	0	0
Theorems	20	53
Proofs	25	63
Total	**60**	**146**

Table 3.1: Summary of Z declarations for Chapter 3.

Declarations	**This Chapter**	**Globally**
Unboxed items	8	14
Axiomatic definitions	1	2
Generic axiomatic defs.	0	1
Schemas	6	13
Generic schemas	0	0
Theorems	20	53
Proofs	25	63
Total	**60**	**146**

Table 3.2: Summary of Z declarations for Chapter 3.

References

Bibliography