Dynamic Architectures

Diego Marmsoler

October 27, 2022

Abstract

The architecture of a system describes the system's overall organization into components and connections between those components. With the emergence of mobile computing, dynamic architectures have become increasingly important. In such architectures, components may appear or disappear, and connections may change over time. In the following we mechanize a theory of dynamic architectures and verify the soundness of a corresponding calculus. Therefore, we first formalize the notion of configuration traces [5] as a model for dynamic architectures. Then, the behavior of single components is formalized in terms of behavior traces and an operator is introduced and studied to extract the behavior of a single component out of a given configuration trace. Then, behavior trace assertions are introduced as a temporal specification technique to specify behavior of components. Reasoning about component behavior in a dynamic context is formalized in terms of a calculus for dynamic architectures [3]. Finally, the soundness of the calculus is verified by introducing an alternative interpretation for behavior trace assertions over configuration traces and proving the rules of the calculus. Since projection may lead to finite as well as infinite behavior traces, they are formalized in terms of coinductive lists. Thus, our theory is based on Lochbihler's [1] formalization of coinductive lists. The theory may be applied to verify properties for dynamic architectures.

Contents

1 A Theory of Dynamic Architectures

The following theory formalizes configuration traces [4, 5] as a model for dynamic architectures. Since configuration traces may be finite as well as infinite, the theory depends on Lochbihler's theory of co-inductive lists [1].

theory *Configuration-Traces*
imports *Coinductive.Coinductive-List*
begin

In the following we first provide some preliminary results for natural numbers, extended natural numbers, and lazy lists. Then, we introduce a locale @textdynamic_architectures which introduces basic definitions and corresponding properties for dynamic architectures.

1.1 Natural Numbers

We provide one additional property for natural numbers.

lemma *boundedGreatest*:
assumes P (i::*nat*)
and $\forall n' > n.\ \neg\ P\ n'$
shows $\exists i' \leq n.\ P\ i' \wedge (\forall n'.\ P\ n' \longrightarrow n' \leq i')$
proof −
have P (i::*nat*) $\Longrightarrow n \geq i \Longrightarrow \forall n' > n.\ \neg\ P\ n' \Longrightarrow (\exists i' \leq n.\ P\ i' \wedge (\forall n' \leq n.\ P\ n' \longrightarrow n' \leq i'))$
proof (*induction n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case*
proof *cases*
assume $i = Suc\ n$
then show *?thesis* **using** *Suc.prems* **by** *auto*
next
assume $\neg(i = Suc\ n)$
thus *?thesis*
proof *cases*
assume P (*Suc n*)
thus *?thesis* **by** *auto*
next
assume $\neg\ P$ (*Suc n*)
with *Suc.prems* **have** $\forall n' > n.\ \neg\ P\ n'$ **using** *Suc-lessI* **by** *blast*
moreover from ‹$\neg(i = Suc\ n)$› **have** $i \leq n$ **and** $P\ i$ **using** *Suc.prems* **by** *auto*
ultimately obtain i' **where** $i' \leq n \wedge P\ i' \wedge (\forall n' \leq n.\ P\ n' \longrightarrow n' \leq i')$ **using** *Suc.IH* **by** *blast*
hence $i' \leq n$ **and** $P\ i'$ **and** $(\forall n' \leq n.\ P\ n' \longrightarrow n' \leq i')$ **by** *auto*
thus *?thesis* **by** (*metis le-SucI le-Suc-eq*)
qed
qed
qed
moreover have $n \geq i$
proof (*rule ccontr*)
assume $\neg\ (n \geq i)$
hence $n < i$ **by** *arith*
thus *False* **using** *assms* **by** *blast*
qed
ultimately obtain i' **where** $i' \leq n$ **and** $P\ i'$ **and** $\forall n' \leq n.\ P\ n' \longrightarrow n' \leq i'$ **using** *assms* **by** *blast*

with *assms* **have** $\forall n'. P\ n' \longrightarrow n' \leq i'$ **using** *not-le-imp-less* **by** *blast*
with ‹$i' \leq n$› **and** ‹$P\ i'$› **show** *?thesis* **by** *auto*
qed

1.2 Extended Natural Numbers

We provide one simple property for the *strict* order over extended natural numbers.

lemma *enat-min*:
assumes $m \geq enat\ n'$
and $enat\ n < m - enat\ n'$
shows $enat\ n + enat\ n' < m$
using *assms* **by** (*metis add.commute enat.simps*(*3*) *enat-add-mono enat-add-sub-same le-iff-add*)

1.3 Lazy Lists

In the following we provide some additional notation and properties for lazy lists.

notation *LNil* ($[]_l$)
notation *LCons* (**infixl** $\#_l$ *60*)
notation *lappend* (**infixl** $@_l$ *60*)

lemma *lnth-lappend*[*simp*]:
assumes *lfinite xs*
and ¬ *lnull ys*
shows *lnth* (*xs* $@_l$ *ys*) (*the-enat* (*llength xs*)) = *lhd ys*
proof −
from *assms* **have** $\exists k.$ *llength xs* = *enat k* **using** *lfinite-conv-llength-enat* **by** *auto*
then obtain *k* **where** *llength xs* = *enat k* **by** *blast*
hence *lnth* (*xs* $@_l$ *ys*) (*the-enat* (*llength xs*)) = *lnth ys 0*
using *lnth-lappend2*[*of xs k k ys*] **by** *simp*
with *assms* **show** *?thesis* **using** *lnth-0-conv-lhd* **by** *simp*
qed

lemma *lfilter-ltake*:
assumes ∀(*n*::*nat*)≤*llength xs*. *n*≥*i* ⟶ (¬ *P* (*lnth xs n*))
shows *lfilter P xs* = *lfilter P* (*ltake i xs*)
proof −
have *lfilter P xs* = *lfilter P* ((*ltake i xs*) $@_l$ (*ldrop i xs*))
using *lappend-ltake-ldrop*[*of* (*enat i*) *xs*] **by** *simp*
hence *lfilter P xs* = (*lfilter P* ((*ltake i*) *xs*)) $@_l$ (*lfilter P* (*ldrop i xs*)) **by** *simp*
show *?thesis*
proof *cases*
assume *enat i* ≤ *llength xs*

have ∀*x*<*llength* (*ldrop i xs*). ¬ *P* (*lnth* (*ldrop i xs*) *x*)
proof (*rule allI*)
fix *x* **show** *enat x* < *llength* (*ldrop* (*enat i*) *xs*) ⟶ ¬ *P* (*lnth* (*ldrop* (*enat i*) *xs*) *x*)
proof
assume *enat x* < *llength* (*ldrop* (*enat i*) *xs*)
moreover have *llength* (*ldrop* (*enat i*) *xs*) = *llength xs* − *enat i*
using *llength-ldrop*[*of enat i*] **by** *simp*
ultimately have *enat x* < *llength xs* − *enat i* **by** *simp*
with ‹*enat i* ≤ *llength xs*› **have** *enat x* + *enat i* < *llength xs*
using *enat-min*[*of i llength xs x*] **by** *simp*
moreover have *enat i* + *enat x* = *enat x* + *enat i* **by** *simp*

```
      ultimately have enat i + enat x < llength xs by arith
      hence i + x < llength xs by simp
      hence lnth (ldrop i xs) x = lnth xs (x + the-enat i) using lnth-ldrop[of enat i x xs] by simp
      moreover have x + i ≥ i by simp
      with assms ‹i + x < llength xs› have ¬ P (lnth xs (x + the-enat i))
        by (simp add: assms(1) add.commute)
      ultimately show ¬ P (lnth (ldrop i xs) x) using assms by simp
    qed
  qed
  hence lfilter P (ldrop i xs) = []l by (metis diverge-lfilter-LNil in-lset-conv-lnth)
  with ‹lfilter P xs = (lfilter P ((ltake i) xs)) @l (lfilter P (ldrop i xs))›
    show lfilter P xs = lfilter P (ltake i xs) by simp
 next
  assume ¬ enat i ≤ llength xs
  hence enat i > llength xs by simp
  hence ldrop i xs = []l by simp
  hence lfilter P (ldrop i xs) = []l using lfilter-LNil[of P] by arith
  with ‹lfilter P xs = (lfilter P ((ltake i) xs)) @l (lfilter P (ldrop i xs))›
    show lfilter P xs = lfilter P (ltake i xs) by simp
 qed
qed

lemma lfilter-lfinite[simp]:
 assumes lfinite (lfilter P t)
  and ¬ lfinite t
 shows ∃ n. ∀ n'>n. ¬ P (lnth t n')
proof −
 from assms have finite {n. enat n < llength t ∧ P (lnth t n)} using lfinite-lfilter by auto
 then obtain k
  where sset: {n. enat n < llength t ∧ P (lnth t n)} ⊆ {n. n<k ∧ enat n < llength t ∧ P (lnth t n)}
  using finite-nat-bounded[of {n. enat n < llength t ∧ P (lnth t n)}] by auto
 show ?thesis
 proof (rule ccontr)
  assume ¬(∃ n. ∀ n'>n. ¬ P (lnth t n'))
  hence ∀ n. ∃ n'>n. P (lnth t n') by simp
  then obtain n' where n'>k and P (lnth t n') by auto
  moreover from ‹¬ lfinite t› have n' < llength t by (simp add: not-lfinite-llength)
  ultimately have n' ∉ {n. n<k ∧ enat n < llength t ∧ P (lnth t n)} and
    n'∈{n. enat n < llength t ∧ P (lnth t n)} by auto
  with sset show False by auto
 qed
qed
```

1.4 Specifying Dynamic Architectures

In the following we formalize dynamic architectures in terms of configuration traces, i.e., sequences of architecture configurations. Moreover, we introduce definitions for operations to support the specification of configuration traces.

```
typedecl cnf
type-synonym trace = nat ⇒ cnf
consts arch:: trace set

type-synonym cta = trace ⇒ nat ⇒ bool
```

1.4.1 Implication

definition *imp* :: *cta* ⇒ *cta* ⇒ *cta* (**infixl** $\longrightarrow^c$ *10*)
  where $\gamma \longrightarrow^c \gamma' \equiv \lambda\ t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$

declare *imp-def*[*simp*]

lemma *impI*[*intro*!]:
  fixes $t\ n$
  assumes $\gamma\ t\ n \Longrightarrow \gamma'\ t\ n$
  shows $(\gamma \longrightarrow^c \gamma')\ t\ n$ **using** *assms* **by** *simp*

lemma *impE*[*elim*!]:
  fixes $t\ n$
  assumes $(\gamma \longrightarrow^c \gamma')\ t\ n$ **and** $\gamma\ t\ n$ **and** $\gamma'\ t\ n \Longrightarrow \gamma''\ t\ n$
  shows $\gamma''\ t\ n$ **using** *assms* **by** *simp*

1.4.2 Disjunction

definition *disj* :: *cta* ⇒ *cta* ⇒ *cta* (**infixl** $\vee^c$ *15*)
  where $\gamma \vee^c \gamma' \equiv \lambda\ t\ n.\ \gamma\ t\ n \vee \gamma'\ t\ n$

declare *disj-def*[*simp*]

lemma *disjI1*[*intro*]:
  assumes $\gamma\ t\ n$
  shows $(\gamma \vee^c \gamma')\ t\ n$ **using** *assms* **by** *simp*

lemma *disjI2*[*intro*]:
  assumes $\gamma'\ t\ n$
  shows $(\gamma \vee^c \gamma')\ t\ n$ **using** *assms* **by** *simp*

lemma *disjE*[*elim*!]:
  assumes $(\gamma \vee^c \gamma')\ t\ n$
    and $\gamma\ t\ n \Longrightarrow \gamma''\ t\ n$
    and $\gamma'\ t\ n \Longrightarrow \gamma''\ t\ n$
  shows $\gamma''\ t\ n$ **using** *assms* **by** *auto*

1.4.3 Conjunction

definition *conj* :: *cta* ⇒ *cta* ⇒ *cta* (**infixl** $\wedge^c$ *20*)
  where $\gamma \wedge^c \gamma' \equiv \lambda\ t\ n.\ \gamma\ t\ n \wedge \gamma'\ t\ n$

declare *conj-def*[*simp*]

lemma *conjI*[*intro*!]:
  fixes n
  assumes $\gamma\ t\ n$ **and** $\gamma'\ t\ n$
  shows $(\gamma \wedge^c \gamma')\ t\ n$ **using** *assms* **by** *simp*

lemma *conjE*[*elim*!]:
  fixes n
  assumes $(\gamma \wedge^c \gamma')\ t\ n$ **and** $\gamma\ t\ n \Longrightarrow \gamma'\ t\ n \Longrightarrow \gamma''\ t\ n$
  shows $\gamma''\ t\ n$ **using** *assms* **by** *simp*

1.4.4 Negation

definition *neg* :: *cta* ⇒ *cta* (¬^{c} - [*19*] *19*)
  where ¬^{c} $\gamma \equiv \lambda\ t\ n.\ \neg\ \gamma\ t\ n$

declare *neg-def*[*simp*]

lemma *negI*[*intro*!]:
  assumes $\gamma\ t\ n \Longrightarrow$ *False*
  shows (¬^{c} γ) $t\ n$ **using** *assms* **by** *auto*

lemma *negE*[*elim*!]:
  assumes (¬^{c} γ) $t\ n$
    and $\gamma\ t\ n$
  shows $\gamma'\ t\ n$ **using** *assms* **by** *simp*

1.4.5 Quantifiers

definition *all* :: ('*a* ⇒ *cta*)
  ⇒ *cta* (**binder** ∀$_c$ *10*)
  where *all* $P \equiv \lambda t\ n.\ (\forall\, y.\ (P\ y\ t\ n))$

declare *all-def*[*simp*]

lemma *allI*[*intro*!]:
  assumes $\bigwedge x.\ \gamma\ x\ t\ n$
  shows (∀$_c x.\ \gamma\ x$) $t\ n$ **using** *assms* **by** *simp*

lemma *allE*[*elim*!]:
  fixes n
  assumes (∀$_c x.\ \gamma\ x$) $t\ n$ **and** $\gamma\ x\ t\ n \Longrightarrow \gamma'\ t\ n$
  shows $\gamma'\ t\ n$ **using** *assms* **by** *simp*

definition *ex* :: ('*a* ⇒ *cta*)
  ⇒ *cta* (**binder** ∃$_c$ *10*)
  where *ex* $P \equiv \lambda t\ n.\ (\exists\, y.\ (P\ y\ t\ n))$

declare *ex-def*[*simp*]

lemma *exI*[*intro*!]:
  assumes $\gamma\ x\ t\ n$
  shows (∃$_c x.\ \gamma\ x$) $t\ n$ **using** *assms HOL.exI* **by** *simp*

lemma *exE*[*elim*!]:
  assumes (∃$_c x.\ \gamma\ x$) $t\ n$ **and** $\bigwedge x.\ \gamma\ x\ t\ n \Longrightarrow \gamma'\ t\ n$
  shows $\gamma'\ t\ n$ **using** *assms HOL.exE* **by** *auto*

1.4.6 Atomic Assertions

First we provide rules for basic behavior assertions.

definition *ca* :: (*cnf* ⇒ *bool*) ⇒ *cta*
  where *ca* $\varphi \equiv \lambda\ t\ n.\ \varphi\ (t\ n)$

lemma *caI*[*intro*]:
  fixes n
  assumes $\varphi\ (t\ n)$

shows $(ca\ \varphi)\ t\ n$ **using** *assms ca-def* **by** *simp*

lemma *caE*[*elim*]:
fixes n
assumes $(ca\ \varphi)\ t\ n$
shows $\varphi\ (t\ n)$ **using** *assms ca-def* **by** *simp*

1.4.7 Next Operator

definition $nxt :: cta \Rightarrow cta$ $(\bigcirc_c(\text{-})\ 24)$
where $\bigcirc_c(\gamma) \equiv \lambda(t{::}(nat \Rightarrow cnf))\ n.\ \gamma\ t\ (Suc\ n)$

1.4.8 Eventually Operator

definition $evt :: cta \Rightarrow cta$ $(\Diamond_c(\text{-})\ 23)$
where $\Diamond_c(\gamma) \equiv \lambda(t{::}(nat \Rightarrow cnf))\ n.\ \exists\, n'{\geq}n.\ \gamma\ t\ n'$

1.4.9 Globally Operator

definition $glob :: cta \Rightarrow cta$ $(\Box_c(\text{-})\ 22)$
where $\Box_c(\gamma) \equiv \lambda(t{::}(nat \Rightarrow cnf))\ n.\ \forall\, n'{\geq}n.\ \gamma\ t\ n'$

lemma *globI*[*intro*!]:
fixes n'
assumes $\forall\, n{\geq}n'.\ \gamma\ t\ n$
shows $(\Box_c(\gamma))\ t\ n'$ **using** *assms glob-def* **by** *simp*

lemma *globE*[*elim*!]:
fixes $n\ n'$
assumes $(\Box_c(\gamma))\ t\ n$ **and** $n'{\geq}n$
shows $\gamma\ t\ n'$ **using** *assms glob-def* **by** *simp*

1.4.10 Until Operator

definition $until :: cta \Rightarrow cta \Rightarrow cta$ (**infixl** $\mathfrak{U}_c$ *21*)
where $\gamma'\ \mathfrak{U}_c\ \gamma \equiv \lambda(t{::}(nat \Rightarrow cnf))\ n.\ \exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma'\ t\ n')$

lemma *untilI*[*intro*]:
fixes n
assumes $\exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n'{<}n'' \longrightarrow \gamma'\ t\ n')$
shows $(\gamma'\ \mathfrak{U}_c\ \gamma)\ t\ n$ **using** *assms until-def* **by** *simp*

lemma *untilE*[*elim*]:
fixes n
assumes $(\gamma'\ \mathfrak{U}_c\ \gamma)\ t\ n$
shows $\exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n'{<}n'' \longrightarrow \gamma'\ t\ n')$ **using** *assms until-def* **by** *simp*

1.4.11 Weak Until

definition $wuntil :: cta \Rightarrow cta \Rightarrow cta$ (**infixl** $\mathfrak{W}_c$ *20*)
where $\gamma'\ \mathfrak{W}_c\ \gamma \equiv \gamma'\ \mathfrak{U}_c\ \gamma \vee^c \Box_c(\gamma')$

lemma *wUntilI*[*intro*]:
fixes n
assumes $(\exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n'{<}n'' \longrightarrow \gamma'\ t\ n')) \vee (\forall\, n'{\geq}n.\ \gamma'\ t\ n')$
shows $(\gamma'\ \mathfrak{W}_c\ \gamma)\ t\ n$ **using** *assms wuntil-def* **by** *auto*

lemma *wUntilE*[*elim*]:
fixes $n\ n'$
assumes $(\gamma'\ \mathfrak{W}_c\ \gamma)\ t\ n$
shows $(\exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n'{<}n'' \longrightarrow \gamma'\ t\ n')) \vee (\forall\, n'{\geq}n.\ \gamma'\ t\ n')$
proof −
from *assms* **have** $(\gamma'\ \mathfrak{U}_c\ \gamma \vee^c \Box_c(\gamma'))\ t\ n$ **using** *wuntil-def* **by** *simp*
hence $(\gamma'\ \mathfrak{U}_c\ \gamma)\ t\ n \vee (\Box_c(\gamma'))\ t\ n$ **by** *simp*
thus *?thesis*
proof
assume $(\gamma'\ \mathfrak{U}_c\ \gamma)\ t\ n$
hence $\exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma'\ t\ n')$ **by** *auto*
thus *?thesis* **by** *auto*
next
assume $(\Box_c\gamma')\ t\ n$
hence $\forall\, n'{\geq}n.\ \gamma'\ t\ n'$ **by** *auto*
thus *?thesis* **by** *auto*
qed
qed

lemma *wUntil-Glob*:
assumes $(\gamma'\ \mathfrak{W}_c\ \gamma)\ t\ n$
and $(\Box_c(\gamma' \longrightarrow^c \gamma''))\ t\ n$
shows $(\gamma''\ \mathfrak{W}_c\ \gamma)\ t\ n$
proof
from *assms*(*1*) **have** $(\exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma'\ t\ n')) \vee (\forall\, n'{\geq}n.\ \gamma'\ t\ n')$ **using** *wUntilE* **by** *simp*
thus $(\exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma''\ t\ n')) \vee (\forall\, n'{\geq}n.\ \gamma''\ t\ n')$
proof
assume $\exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma'\ t\ n')$
show $(\exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma''\ t\ n')) \vee (\forall\, n'{\geq}n.\ \gamma''\ t\ n')$
proof −
from ‹$\exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma'\ t\ n')$› **obtain** n'' **where** $n''{\geq}n$ **and** $\gamma\ t\ n''$ **and** *a1*: $\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma'\ t\ n'$ **by** *auto*
moreover have $\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma''\ t\ n'$
proof
fix n'
show $n'{\geq}n \longrightarrow n'{<}\ n'' \longrightarrow \gamma''\ t\ n'$
proof (*rule HOL.impI*[*OF HOL.impI*])
assume $n'{\geq}n$ **and** $n'{<}n''$
with *assms*(*2*) **have** $(\gamma' \longrightarrow^c \gamma'')\ t\ n'$ **using** *globE* **by** *simp*
hence $\gamma'\ t\ n' \longrightarrow \gamma''\ t\ n'$ **using** *impE* **by** *auto*
moreover from *a1* ‹$n'{\geq}n$› ‹$n'{<}n''$› **have** $\gamma'\ t\ n'$ **by** *simp*
ultimately show $\gamma''\ t\ n'$ **by** *simp*
qed
qed
ultimately show *?thesis* **by** *auto*
qed
next
assume *a1*: $\forall\, n'{\geq}n.\ \gamma'\ t\ n'$
have $\forall\, n'{\geq}n.\ \gamma''\ t\ n'$
proof
fix n'
show $n'{\geq}n \longrightarrow \gamma''\ t\ n'$
proof

```
      assume n'≥n
      with assms(2) have (γ' ⟶^c γ'') t n' using globE by simp
      hence γ' t n' ⟶ γ'' t n' using impE by auto
      moreover from a1 ‹n'≥n› have γ' t n' by simp
      ultimately show γ'' t n' by simp
    qed
  qed
  thus (∃n''≥n. γ t n'' ∧ (∀n'≥n. n' < n'' ⟶ γ'' t n')) ∨ (∀n'≥n. γ'' t n') by simp
 qed
qed
```

1.5 Dynamic Components

To support the specification of patterns over dynamic architectures we provide a locale for dynamic components. It takes the following type parameters:

- id: a type for component identifiers
- cmp: a type for components
- cnf: a type for architecture configurations

```
locale dynamic-component =
 fixes tCMP :: 'id ⇒ cnf ⇒ 'cmp (σ_(-) [0,110]60)
   and active :: 'id ⇒ cnf ⇒ bool (‖-‖_ [0,110]60)
begin
```

The locale requires two parameters:

- *tCMP* is an operator to obtain a component with a certain identifier from an architecture configuration.
- *active* is a predicate to assert whether a certain component is activated within an architecture configuration.

The locale provides some general properties about its parameters and introduces six important operators over configuration traces:

- An operator to extract the behavior of a certain component out of a given configuration trace.
- An operator to obtain the number of activations of a certain component within a given configuration trace.
- An operator to obtain the least point in time (before a certain point in time) from which on a certain component is not activated anymore.
- An operator to obtain the latest point in time where a certain component was activated.
- Two operators to map time-points between configuration traces and behavior traces.

Moreover, the locale provides several properties about the operators and their relationships.

```
lemma nact-active:
 fixes t::nat ⇒ cnf
```

and n::nat
and n''
and id
assumes $\|id\|_{t\ n}$
and $n'' \geq n$
and $\neg\ (\exists\, n'{\geq}n.\ n' < n'' \wedge \|id\|_{t\ n'})$
shows $n{=}n''$
using *assms le-eq-less-or-eq* **by** *auto*

lemma *nact-exists*:
fixes t::$nat \Rightarrow cnf$
assumes $\exists\, i{\geq}n.\ \|c\|_{t\ i}$
shows $\exists\, i{\geq}n.\ \|c\|_{t\ i} \wedge (\nexists k.\ n{\leq}k \wedge k{<}i \wedge \|c\|_{t\ k})$
proof −
let *?L* = *LEAST i*. $(i{\geq}n \wedge \|c\|_{t\ i})$
from *assms* **have** $?L{\geq}n \wedge \|c\|_{t\ ?L}$ **using** *LeastI*[*of* λx::*nat*. $(x{\geq}n \wedge \|c\|_{t\ x})$] **by** *auto*
moreover have $\nexists k.\ n{\leq}k \wedge k{<}?L \wedge \|c\|_{t\ k}$ **using** *not-less-Least* **by** *auto*
ultimately show *?thesis* **by** *blast*
qed

lemma *lActive-least*:
assumes $\exists\, i{\geq}n.\ i < llength\ t \wedge \|c\|_{lnth\ t\ i}$
shows $\exists\, i{\geq}n.\ (i < llength\ t \wedge \|c\|_{lnth\ t\ i} \wedge (\nexists k.\ n{\leq}k \wedge k{<}i \wedge k{<}llength\ t \wedge \|c\|_{lnth\ t\ k}))$
proof −
let *?L* = *LEAST i*. $(i{\geq}n \wedge i < llength\ t \wedge \|c\|_{lnth\ t\ i})$
from *assms* **have** $?L{\geq}n \wedge ?L < llength\ t \wedge \|c\|_{lnth\ t\ ?L}$
using *LeastI*[*of* λx::*nat*.$(x{\geq}n \wedge x{<}llength\ t \wedge \|c\|_{lnth\ t\ x})$] **by** *auto*
moreover have $\nexists k.\ n{\leq}k \wedge k{<}llength\ t \wedge k{<}?L \wedge \|c\|_{lnth\ t\ k}$ **using** *not-less-Least* **by** *auto*
ultimately show *?thesis* **by** *blast*
qed

1.6 Projection

In the following we introduce an operator which extracts the behavior of a certain component out of a given configuration trace.

definition *proj*:: *'id* $\Rightarrow$ (*cnf llist*) $\Rightarrow$ (*'cmp llist*) ($\pi_{\text{-}}$(-) [*0*,*110*]*60*)
where *proj c* = *lmap* $(\lambda cnf.\ (\sigma_c(cnf)))$ $\circ$ (*lfilter* (*active c*))

lemma *proj-lnil* [*simp*,*intro*]:
$\pi_c([]_l) = []_l$ **using** *proj-def* **by** *simp*

lemma *proj-lnull* [*simp*]:
$\pi_c(t) = []_l \longleftrightarrow (\forall\, k \in lset\ t.\ \neg\ \|c\|_k)$
proof
assume $\pi_c(t) = []_l$
hence *lfilter* (*active c*) $t = []_l$ **using** *proj-def lmap-eq-LNil* **by** *auto*
thus $\forall\, k \in lset\ t.\ \neg\ \|c\|_k$ **using** *lfilter-eq-LNil*[*of active c*] **by** *simp*
next
assume $\forall\, k{\in}lset\ t.\ \neg\ \|c\|_k$
hence *lfilter* (*active c*) $t = []_l$ **by** *simp*
thus $\pi_c(t) = []_l$ **using** *proj-def* **by** *simp*
qed

lemma *proj-LCons* [*simp*]:
$\pi_i(x\ \#_l\ xs) = (if\ \|i\|_x\ then\ (\sigma_i(x))\ \#_l\ (\pi_i(xs))\ else\ \pi_i(xs))$

using *proj-def* **by** *simp*

lemma *proj-llength*[*simp*]:
llength ($\pi_c(t)$) $\leq$ *llength* t
using *llength-lfilter-ile proj-def* **by** *simp*

lemma *proj-ltake*:
assumes $\forall (n'::nat) \leq llength\ t.\ n' \geq n \longrightarrow (\neg \|c\|_{lnth\ t\ n'})$
shows $\pi_c(t) = \pi_c(ltake\ n\ t)$ **using** *lfilter-ltake proj-def assms* **by** (*metis comp-apply*)

lemma *proj-finite-bound*:
assumes *lfinite* (π_c(*inf-llist* t))
shows $\exists\, n.\ \forall\, n' > n.\ \neg\ \|c\|_{t\ n'}$
using *assms lfilter-lfinite*[*of active c inf-llist t*] *proj-def* **by** *simp*

1.6.1 Monotonicity and Continuity

lemma *proj-mcont*:
shows *mcont lSup lprefix lSup lprefix* (*proj c*)
proof −
have *mcont lSup lprefix lSup lprefix* ($\lambda x.$ *lmap* ($\lambda cnf.$ $\sigma_c(cnf)$) (*lfilter* (*active c*) x))
by *simp*
moreover have ($\lambda x.$ *lmap* ($\lambda cnf.$ $\sigma_c(cnf)$) (*lfilter* (*active c*) x)) =
lmap ($\lambda cnf.$ $\sigma_c(cnf)$) $\circ$ *lfilter* (*active c*) **by** *auto*
ultimately show *?thesis* **using** *proj-def* **by** *simp*
qed

lemma *proj-mcont2mcont*:
assumes *mcont lub ord lSup lprefix f*
shows *mcont lub ord lSup lprefix* ($\lambda x.$ $\pi_c(f\ x)$)
proof −
have *mcont lSup lprefix lSup lprefix* (*proj c*) **using** *proj-mcont* **by** *simp*
with *assms* **show** *?thesis* **using** *llist.mcont2mcont*[*of lSup lprefix proj c*] **by** *simp*
qed

lemma *proj-mono-prefix*[*simp*]:
assumes *lprefix t t'*
shows *lprefix* ($\pi_c(t)$) ($\pi_c(t')$)
proof −
from *assms* **have** *lprefix* (*lfilter* (*active c*) *t*) (*lfilter* (*active c*) *t'*) **using** *lprefix-lfilterI* **by** *simp*
hence *lprefix* (*lmap* ($\lambda cnf.$ $\sigma_c(cnf)$) (*lfilter* (*active c*) *t*))
(*lmap* ($\lambda cnf.$ $\sigma_c(cnf)$) (*lfilter* (*active c*) *t'*)) **using** *lmap-lprefix* **by** *simp*
thus *?thesis* **using** *proj-def* **by** *simp*
qed

1.6.2 Finiteness

lemma *proj-finite*[*simp*]:
assumes *lfinite t*
shows *lfinite* ($\pi_c(t)$)
using *assms proj-def* **by** *simp*

lemma *proj-finite2*:
assumes $\forall (n'::nat) \leq llength\ t.\ n' \geq n \longrightarrow (\neg\ \|c\|_{lnth\ t\ n'})$
shows *lfinite* ($\pi_c(t)$) **using** *assms proj-ltake proj-finite* **by** *simp*

lemma *proj-append-lfinite*[*simp*]:
fixes $t\ t'$
assumes *lfinite t*
shows $\pi_c(t\ @_l\ t') = (\pi_c(t))\ @_l\ (\pi_c(t'))$ (**is** *?lhs=?rhs*)
proof −
have *?lhs* = (*lmap* ($\lambda cnf.\ \sigma_c(cnf)$) ∘ (*lfilter* (*active c*))) (*t* $@_l$ *t'*) **using** *proj-def* **by** *simp*
also have ... = *lmap* ($\lambda cnf.\ \sigma_c(cnf)$) (*lfilter* (*active c*) (*t* $@_l$ *t'*)) **by** *simp*
also from *assms* **have** ... = *lmap* ($\lambda cnf.\ \sigma_c(cnf)$)
((*lfilter* (*active c*) *t*) $@_l$ (*lfilter* (*active c*) *t'*)) **by** *simp*
also have ... = ($@_l$) (*lmap* ($\lambda cnf.\ \sigma_c(cnf)$) (*lfilter* (*active c*) *t*))
(*lmap* ($\lambda cnf.\ \sigma_c(cnf)$) (*lfilter* (*active c*) *t'*)) **using** *lmap-lappend-distrib* **by** *simp*
also have ... = *?rhs* **using** *proj-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *proj-one*:
assumes $\exists\, i.\ i < llength\ t \wedge \|c\|_{lnth\ t\ i}$
shows *llength* ($\pi_c(t)$) ≥ *1*
proof −
from *assms* **have** $\exists\, x{\in}lset\ t.\ \|c\|_x$ **using** *lset-conv-lnth* **by** *force*
hence ¬ *lfilter* ($\lambda k.\ \|c\|_k$) *t* = $[]_l$ **using** *lfilter-eq-LNil*[*of* ($\lambda k.\ \|c\|_k$)] **by** *blast*
hence ¬ $\pi_c(t) = []_l$ **using** *proj-def* **by** *fastforce*
thus *?thesis* **by** (*simp add*: *ileI1 lnull-def one-eSuc*)
qed

1.6.3 Projection not Active

lemma *proj-not-active*[*simp*]:
assumes *enat n* < *llength t*
and ¬ $\|c\|_{lnth\ t\ n}$
shows $\pi_c(ltake\ (Suc\ n)\ t) = \pi_c(ltake\ n\ t)$ (**is** *?lhs = ?rhs*)
proof −
from *assms* **have** *ltake* (*enat* (*Suc n*)) *t* = (*ltake* (*enat n*) *t*) $@_l$ ((*lnth t n*) $\#_l$ $[]_l$)
using *ltake-Suc-conv-snoc-lnth* **by** *blast*
hence *?lhs* = π_c((*ltake* (*enat n*) *t*) $@_l$ ((*lnth t n*) $\#_l$ $[]_l$)) **by** *simp*
moreover have ... = (π_c(*ltake* (*enat n*) *t*)) $@_l$ (π_c((*lnth t n*) $\#_l$ $[]_l$)) **by** *simp*
moreover from *assms* **have** π_c((*lnth t n*) $\#_l$ $[]_l$) = $[]_l$ **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *proj-not-active-same*:
assumes *enat n* ≤ (*n'*::*enat*)
and ¬ *lfinite t* ∨ *n'*−*1* < *llength t*
and ∄ *k*. *k*≥*n* ∧ *k*<*n'* ∧ *k* < *llength t* ∧ $\|c\|_{lnth\ t\ k}$
shows $\pi_c(ltake\ n'\ t) = \pi_c(ltake\ n\ t)$
proof −
have $\pi_c(ltake\ (n + (n' - n))\ t) = \pi_c((ltake\ n\ t)\ @_l\ (ltake\ (n'{-}n)\ (ldrop\ n\ t)))$
by (*simp add*: *ltake-plus-conv-lappend*)
hence $\pi_c(ltake\ (n + (n' - n))\ t) =$
(π_c(*ltake n t*)) $@_l$ (π_c(*ltake* (*n'*−*n*) (*ldrop n t*))) **by** *simp*
moreover have $\pi_c(ltake\ (n'{-}n)\ (ldrop\ n\ t)) = []_l$
proof −
have ∀ *k*∈{*lnth* (*ltake* (*n'* − *enat n*) (*ldrop* (*enat n*) *t*)) *na* |
na. *enat na* < *llength* (*ltake* (*n'* − *enat n*) (*ldrop* (*enat n*) *t*))}. ¬ $\|c\|_k$
proof
fix *k* **assume** *k*∈{*lnth* (*ltake* (*n'* − *enat n*) (*ldrop* (*enat n*) *t*)) *na* |

na. enat na < llength (ltake (n′ − enat n) (ldrop (enat n) t))}
then obtain *k′* **where** *enat k′ < llength (ltake (n′ − enat n) (ldrop (enat n) t))*
and *k=lnth (ltake (n′ − enat n) (ldrop (enat n) t)) k′* **by** *auto*
have *enat (k′ + n) < llength t*
proof −
from ‹*enat k′ < llength (ltake (n′ − enat n) (ldrop (enat n) t))*› **have** *enat k′ < n′−n* **by** *simp*
hence *enat k′ + n < n′* **using** *assms(1) enat-min* **by** *auto*
show *?thesis*
proof *cases*
assume *lfinite t*
with ‹¬ *lfinite t ∨ n′−1 < llength t*› **have** *n′−1<llength t* **by** *simp*
hence *n′< eSuc (llength t)* **by** (*metis eSuc-minus-1 enat-minus-mono1 leD leI*)
hence *n′≤ llength t* **using** *eSuc-ile-mono ileI1* **by** *blast*
with ‹*enat k′ + n < n′*› **show** *?thesis* **by** (*simp add*: *add.commute*)
next
assume ¬ *lfinite t*
hence *llength t = ∞* **using** *not-lfinite-llength* **by** *auto*
thus *?thesis* **by** *simp*
qed
qed
moreover have *k = lnth t (k′ + n)*
proof −
from ‹*enat k′ < llength (ltake (n′ − enat n) (ldrop (enat n) t))*›
have *enat k′<n′ − enat n* **by** *auto*
hence *lnth (ltake (n′ − enat n) (ldrop (enat n) t)) k′ = lnth (ldrop (enat n) t) k′*
using *lnth-ltake*[*of k′ n′ − enat n*] **by** *simp*
with ‹*enat (k′ + n) < llength t*› **show** *?thesis* **using** *lnth-ldrop*[*of n k′ t*]
using ‹*k = lnth (ltake (n′ − enat n) (ldrop (enat n) t)) k′*› **by** (*simp add*: *add.commute*)
qed
moreover from ‹*enat n ≤ (n′::enat)*› **have** *k′ + the-enat n≥n* **by** *auto*
moreover from ‹*enat k′ < llength (ltake (n′ − enat n) (ldrop (enat n) t))*› **have** *k′ + n<n′*
using *assms(1) enat-min* **by** *auto*
ultimately show ¬ $\|c\|_k$ **using** ‹$\nexists$ *k. k≥n ∧ k<n′ ∧ k < llength t ∧* $\|c\|_{lnth\ t\ k}$›
by *simp*
qed
hence ∀ *k∈lset (ltake (n′−n) (ldrop n t))*. ¬ $\|c\|_k$
using *lset-conv-lnth*[*of (ltake (n′ − enat n) (ldrop (enat n) t))*] **by** *simp*
thus *?thesis* **using** *proj-lnull* **by** *auto*
qed
moreover from *assms* **have** *n + (n′ − n) = n′*
by (*meson enat.distinct(1) enat-add-sub-same enat-diff-cancel-left enat-le-plus-same(1) less-imp-le*)
ultimately show *?thesis* **by** *simp*
qed

1.6.4 Projection Active

lemma *proj-active*[*simp*]:
assumes *enat i < llength t* $\|c\|_{lnth\ t\ i}$
shows π_c(*ltake (Suc i) t*) = (π_c(*ltake i t*)) @$_l$ ((σ_c(*lnth t i*)) #$_l$ []$_l$) (**is** *?lhs = ?rhs*)
proof −
from *assms* **have** *ltake (enat (Suc i)) t = (ltake (enat i) t) @$_l$ ((lnth t i) #$_l$ []$_l$)*
using *ltake-Suc-conv-snoc-lnth* **by** *blast*
hence *?lhs =* π_c(*(ltake (enat i) t) @$_l$ ((lnth t i) #$_l$ []$_l$)*) **by** *simp*
moreover have *... =* (π_c(*ltake (enat i) t*)) @$_l$ (π_c(*(lnth t i) #$_l$ []$_l$*)) **by** *simp*
moreover from *assms* **have** π_c(*(lnth t i) #$_l$ []$_l$*) = (σ_c(*lnth t i*)) #$_l$ []$_l$ **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *proj-active-append*:
assumes *a1*: $(n::nat) \leq i$
and *a2*: $enat\ i < (n'::enat)$
and *a3*: $\neg\ lfinite\ t \vee n'-1 < llength\ t$
and *a4*: $\|c\|_{lnth\ t\ i}$
and $\forall i'.\ (n \leq i' \wedge enat\ i' < n' \wedge i' < llength\ t \wedge \|c\|_{lnth\ t\ i'}) \longrightarrow (i' = i)$
shows $\pi_c(ltake\ n'\ t) = (\pi_c(ltake\ n\ t))\ @_l\ ((\sigma_c(lnth\ t\ i))\ \#_l\ []_l)$ (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $\pi_c(ltake\ (Suc\ i)\ t)$
proof −
from *a2* **have** $Suc\ i \leq n'$ **by** (*simp add*: *Suc-ile-eq*)
moreover from *a3* **have** $\neg\ lfinite\ t \vee n'-1 < llength\ t$ **by** *simp*
moreover have $\nexists k.\ enat\ k \geq enat\ (Suc\ i) \wedge k < n' \wedge k < llength\ t \wedge \|c\|_{lnth\ t\ k}$
proof
assume $\exists k.\ enat\ k \geq enat\ (Suc\ i) \wedge k < n' \wedge k < llength\ t \wedge \|c\|_{lnth\ t\ k}$
then obtain *k* **where** $enat\ k \geq enat\ (Suc\ i)$ **and** $k < n'$ **and** $k < llength\ t$ **and** $\|c\|_{lnth\ t\ k}$ **by** *blast*
moreover from ‹$enat\ k \geq enat\ (Suc\ i)$› **have** $enat\ k \geq n$
using *assms* **by** (*meson dual-order.trans enat-ord-simps*(*1*) *le-SucI*)
ultimately have $enat\ k = enat\ i$ **using** *assms* **using** *enat-ord-simps*(*1*) **by** *blast*
with ‹$enat\ k \geq enat\ (Suc\ i)$› **show** *False* **by** *simp*
qed
ultimately show *?thesis* **using** *proj-not-active-same*[*of Suc i n' t c*] **by** *simp*
qed
also have ... = $(\pi_c(ltake\ i\ t))\ @_l\ ((\sigma_c(lnth\ t\ i))\ \#_l\ []_l)$
proof −
have $i < llength\ t$
proof *cases*
assume *lfinite t*
with *a3* **have** $n'-1 < llength\ t$ **by** *simp*
hence $n' \leq llength\ t$ **by** (*metis eSuc-minus-1 enat-minus-mono1 ileI1 not-le*)
with *a2* **show** $enat\ i < llength\ t$ **by** *simp*
next
assume $\neg\ lfinite\ t$
thus *?thesis* **by** (*metis enat-ord-code*(*4*) *llength-eq-infty-conv-lfinite*)
qed
with *a4* **show** *?thesis* **by** *simp*
qed
also have ... = *?rhs*
proof −
from *a1* **have** $enat\ n \leq enat\ i$ **by** *simp*
moreover from *a2 a3* **have** $\neg\ lfinite\ t \vee enat\ i-1 < llength\ t$
using *enat-minus-mono1 less-imp-le order.strict-trans1* **by** *blast*
moreover have $\nexists k.\ k \geq n \wedge enat\ k < enat\ i \wedge enat\ k < llength\ t \wedge \|c\|_{lnth\ t\ k}$
proof
assume $\exists k.\ k \geq n \wedge enat\ k < enat\ i \wedge enat\ k < llength\ t \wedge \|c\|_{lnth\ t\ k}$
then obtain *k* **where** $k \geq n$ **and** $enat\ k < enat\ i$ **and** $enat\ k < llength\ t$ **and** $\|c\|_{lnth\ t\ k}$ **by** *blast*
moreover from ‹$enat\ k < enat\ i$› **have** $enat\ k < n'$ **using** *assms dual-order.strict-trans* **by** *blast*
ultimately have $enat\ k = enat\ i$ **using** *assms* **by** *simp*
with ‹$enat\ k < enat\ i$› **show** *False* **by** *simp*
qed
ultimately show *?thesis* **using** *proj-not-active-same*[*of n i t c*] **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed

1.6.5 Same and not Same

lemma *proj-same-not-active*:
assumes $n \leq n'$
and *enat* $(n'-1) <$ *llength* t
and $\pi_c(ltake\ n'\ t) = \pi_c(ltake\ n\ t)$
shows $\nexists k.\ k \geq n \wedge k < n' \wedge \|c\|_{lnth\ t\ k}$
proof
assume $\exists k.\ k \geq n \wedge k < n' \wedge \|c\|_{lnth\ t\ k}$
then obtain i **where** $i \geq n$ **and** $i < n'$ **and** $\|c\|_{lnth\ t\ i}$ **by** *blast*
moreover from ‹*enat* $(n'-1) <$ *llength* t› **and** ‹$i < n'$› **have** $i <$ *llength* t
by (*metis diff-Suc-1 dual-order.strict-trans enat-ord-simps*(2) *lessE*)
ultimately have $\pi_c(ltake\ (Suc\ i)\ t) =$
$(\pi_c(ltake\ i\ t)) \ @_l\ ((\sigma_c(lnth\ t\ i)) \ \#_l\ []_l)$ **by** *simp*
moreover from ‹$i < n'$› **have** *Suc* $i \leq n'$ **by** *simp*
hence *lprefix*$(\pi_c(ltake\ (Suc\ i)\ t))\ (\pi_c(ltake\ n'\ t))$ **by** *simp*
then obtain *tl* **where** $\pi_c(ltake\ n'\ t) = (\pi_c(ltake\ (Suc\ i)\ t)) \ @_l\ tl$
using *lprefix-conv-lappend* **by** *auto*
moreover from ‹$n \leq i$› **have** *lprefix*$(\pi_c(ltake\ n\ t))\ (\pi_c(ltake\ i\ t))$ **by** *simp*
hence *lprefix*$(\pi_c(ltake\ n\ t))\ (\pi_c(ltake\ i\ t))$ **by** *simp*
then obtain *hd* **where** $\pi_c(ltake\ i\ t) = (\pi_c(ltake\ n\ t)) \ @_l\ hd$
using *lprefix-conv-lappend* **by** *auto*
ultimately have $\pi_c(ltake\ n'\ t) =$
$(((\pi_c(ltake\ n\ t)) \ @_l\ hd) \ @_l\ ((\sigma_c(lnth\ t\ i)) \ \#_l\ []_l)) \ @_l\ tl$ **by** *simp*
also have $\ldots = ((\pi_c(ltake\ n\ t)) \ @_l\ hd) \ @_l\ ((\sigma_c(lnth\ t\ i)) \ \#_l\ tl)$
using *lappend-snocL1-conv-LCons2*[*of* $(\pi_c(ltake\ n\ t)) \ @_l\ hd\ \sigma_c(lnth\ t\ i)$] **by** *simp*
also have $\ldots = (\pi_c(ltake\ n\ t)) \ @_l\ (hd \ @_l\ ((\sigma_c(lnth\ t\ i)) \ \#_l\ tl))$
using *lappend-assoc* **by** *auto*
also have $\pi_c(ltake\ n'\ t) = (\pi_c(ltake\ n'\ t)) \ @_l\ []_l$ **by** *simp*
finally have $(\pi_c(ltake\ n'\ t)) \ @_l\ []_l = (\pi_c(ltake\ n\ t)) \ @_l\ (hd \ @_l\ ((\sigma_c(lnth\ t\ i)) \ \#_l\ tl))$ **.**
moreover from *assms*(3) **have** *llength* $(\pi_c(ltake\ n'\ t)) =$ *llength* $(\pi_c(ltake\ n\ t))$ **by** *simp*
ultimately have *lfinite* $(\pi_c(ltake\ n'\ t)) \longrightarrow []_l = hd \ @_l\ ((\sigma_c(lnth\ t\ i)) \ \#_l\ tl)$
using *assms*(3) *lappend-eq-lappend-conv*[*of* $\pi_c(ltake\ n'\ t)\ \pi_c(ltake\ n\ t)\ []_l$] **by** *simp*
moreover have *lfinite* $(\pi_c(ltake\ n'\ t))$ **by** *simp*
ultimately have $[]_l = hd \ @_l\ ((\sigma_c(lnth\ t\ i)) \ \#_l\ tl)$ **by** *simp*
hence $(\sigma_c(lnth\ t\ i)) \ \#_l\ tl = []_l$ **using** *LNil-eq-lappend-iff* **by** *auto*
thus *False* **by** *simp*
qed

lemma *proj-not-same-active*:
assumes *enat* $n \leq (n'::enat)$
and $(\neg$ *lfinite* $t) \vee n'-1 <$ *llength* t
and $\neg(\pi_c(ltake\ n'\ t) = \pi_c(ltake\ n\ t))$
shows $\exists k.\ k \geq n \wedge k < n' \wedge$ *enat* $k <$ *llength* $t \wedge \|c\|_{lnth\ t\ k}$
proof (*rule ccontr*)
assume $\neg(\exists k.\ k \geq n \wedge k < n' \wedge$ *enat* $k <$ *llength* $t \wedge \|c\|_{lnth\ t\ k})$
have $\pi_c(ltake\ n'\ t) = \pi_c(ltake\ (enat\ n)\ t)$
proof *cases*
assume *lfinite* t
hence *llength* $t \neq \infty$ **by** (*simp add*: *lfinite-llength-enat*)
hence *enat* (*the-enat* (*llength* t)) = *llength* t **by** *auto*
with *assms* ‹$\neg$ $(\exists k \geq n.\ k < n' \wedge$ *enat* $k <$ *llength* $t \wedge \|c\|_{lnth\ t\ k})$›
show *?thesis* **using** *proj-not-active-same*[*of* $n\ n'\ t\ c$] **by** *simp*
next
assume $\neg$ *lfinite* t
with *assms* ‹$\neg$ $(\exists k \geq n.\ k < n' \wedge$ *enat* $k <$ *llength* $t \wedge \|c\|_{lnth\ t\ k})$›

show *?thesis* **using** *proj-not-active-same*[*of n n' t c*] **by** *simp*
qed
with *assms* **show** *False* **by** *simp*
qed

1.7 Activations

We also introduce an operator to obtain the number of activations of a certain component within a given configuration trace.

definition *nAct* :: *'id* ⇒ *enat* ⇒ (*cnf llist*) ⇒ *enat* (⟨- #_ -⟩) **where**
⟨*c* #*n* *t*⟩ ≡ *llength* (*π_c*(*ltake n t*))

lemma *nAct-0*[*simp*]:
⟨*c* #*0* *t*⟩ = *0* **by** (*simp add*: *nAct-def*)

lemma *nAct-NIL*[*simp*]:
⟨*c* #*n* []*l*⟩ = *0* **by** (*simp add*: *nAct-def*)

lemma *nAct-Null*:
assumes *llength t* ≥ *n*
and ⟨*c* #*n* *t*⟩ = *0*
shows ∀ *i*<*n*. ¬ ‖*c*‖*lnth t i*
proof −
from *assms* **have** *lnull* (*π_c*(*ltake n t*)) **using** *nAct-def lnull-def* **by** *simp*
hence *π_c*(*ltake n t*) = []*l* **using** *lnull-def* **by** *blast*
hence (∀ *k*∈*lset* (*ltake n t*). ¬ ‖*c*‖*k*) **by** *simp*
show *?thesis*
proof (*rule ccontr*)
assume ¬ (∀ *i*<*n*. ¬ ‖*c*‖*lnth t i*)
then obtain *i* **where** *i*<*n* **and** ‖*c*‖*lnth t i* **by** *blast*
moreover have *enat i* < *llength* (*ltake n t*) ∧ *lnth* (*ltake n t*) *i* = (*lnth t i*)
proof
from ‹*llength t* ≥ *n*› **have** *n* = *min n* (*llength t*) **using** *min.orderE* **by** *auto*
hence *llength* (*ltake n t*) = *n* **by** *simp*
with ‹*i*<*n*› **show** *enat i* < *llength* (*ltake n t*) **by** *auto*
from ‹*i*<*n*› **show** *lnth* (*ltake n t*) *i* = (*lnth t i*) **using** *lnth-ltake* **by** *auto*
qed
hence (*lnth t i* ∈ *lset* (*ltake n t*)) **using** *in-lset-conv-lnth*[*of lnth t i ltake n t*] **by** *blast*
ultimately show *False* **using** ‹(∀ *k*∈*lset* (*ltake n t*). ¬ ‖*c*‖*k*)› **by** *simp*
qed
qed

lemma *nAct-ge-one*[*simp*]:
assumes *llength t* ≥ *n*
and *i* < *n*
and ‖*c*‖*lnth t i*
shows ⟨*c* #*n* *t*⟩ ≥ *enat 1*
proof (*rule ccontr*)
assume ¬ (⟨*c* #*n* *t*⟩ ≥ *enat 1*)
hence ⟨*c* #*n* *t*⟩ < *enat 1* **by** *simp*
hence ⟨*c* #*n* *t*⟩ < *1* **using** *enat-1* **by** *simp*
hence ⟨*c* #*n* *t*⟩ = *0* **using** *Suc-ile-eq* ‹¬ *enat 1* ≤ ⟨*c* #*n* *t*⟩› *zero-enat-def* **by** *auto*
with ‹*llength t* ≥ *n*› **have** ∀ *i*<*n*. ¬ ‖*c*‖*lnth t i* **using** *nAct-Null* **by** *simp*
with *assms* **show** *False* **by** *simp*
qed

lemma *nAct-finite*[*simp*]:
assumes $n \neq \infty$
shows $\exists\, n'.\ \langle c\ \#_n\ t\rangle = enat\ n'$
proof −
from *assms* **have** *lfinite* (*ltake n t*) **by** *simp*
hence *lfinite* (π_c(*ltake n t*)) **by** *simp*
hence $\exists\, n'.$ *llength* (π_c(*ltake n t*)) = *enat* n' **using** *lfinite-llength-enat*[*of* π_c(*ltake n t*)] **by** *simp*
thus *?thesis* **using** *nAct-def* **by** *simp*
qed

lemma *nAct-enat-the-nat*[*simp*]:
assumes $n \neq \infty$
shows *enat* (*the-enat* ($\langle c\ \#_n\ t\rangle$)) = $\langle c\ \#_n\ t\rangle$
proof −
from *assms* **have** $\langle c\ \#_n\ t\rangle \neq \infty$ **by** *simp*
thus *?thesis* **using** *enat-the-enat* **by** *simp*
qed

1.7.1 Monotonicity and Continuity

lemma *nAct-mcont*:
shows *mcont lSup lprefix Sup* (≤) (*nAct c n*)
proof −
have *mcont lSup lprefix lSup lprefix* (*ltake n*) **by** *simp*
hence *mcont lSup lprefix lSup lprefix* (λ*t*. π_c(*ltake n t*))
using *proj-mcont2mcont*[*of lSup lprefix* (*ltake n*)] **by** *simp*
hence *mcont lSup lprefix Sup* (≤) (λ*t*. *llength* (π_c(*ltake n t*))) **by** *simp*
moreover have *nAct c n* = (λ*t*. *llength* (π_c(*ltake n t*))) **using** *nAct-def* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

lemma *nAct-mono*:
assumes $n \leq n'$
shows $\langle c\ \#_n\ t\rangle \leq \langle c\ \#_{n'}\ t\rangle$
proof −
from *assms* **have** *lprefix* (*ltake n t*) (*ltake n' t*) **by** *simp*
hence *lprefix* (π_c(*ltake n t*)) (π_c(*ltake n' t*)) **by** *simp*
hence *llength* (π_c(*ltake n t*)) ≤ *llength* (π_c(*ltake n' t*))
using *lprefix-llength-le*[*of* (π_c(*ltake n t*))] **by** *simp*
thus *?thesis* **using** *nAct-def* **by** *simp*
qed

lemma *nAct-strict-mono-back*:
assumes $\langle c\ \#_n\ t\rangle < \langle c\ \#_{n'}\ t\rangle$
shows $n < n'$
proof (*rule ccontr*)
assume ¬ $n{<}n'$
hence $n{\geq}n'$ **by** *simp*
hence $\langle c\ \#_n\ t\rangle \geq \langle c\ \#_{n'}\ t\rangle$ **using** *nAct-mono* **by** *simp*
thus *False* **using** *assms* **by** *simp*
qed

1.7.2 Not Active

lemma *nAct-not-active*[*simp*]:

fixes n::nat
and n'::nat
and t::$(cnf\ llist)$
and c::$'id$
assumes $enat\ i < llength\ t$
and $\neg \|c\|_{lnth\ t\ i}$
shows $\langle c\ \#_{Suc\ i}\ t\rangle = \langle c\ \#_i\ t\rangle$
proof −
from *assms* **have** $\pi_c(ltake\ (Suc\ i)\ t) = \pi_c(ltake\ i\ t)$ **by** *simp*
hence $llength\ (\pi_c(ltake\ (enat\ (Suc\ i))\ t)) = llength\ (\pi_c(ltake\ i\ t))$ **by** *simp*
moreover have $llength\ (\pi_c(ltake\ i\ t)) \neq \infty$
using *llength-eq-infty-conv-lfinite*[*of* $\pi_c(ltake\ (enat\ i)\ t)$] **by** *simp*
ultimately have $llength\ (\pi_c(ltake\ (Suc\ i)\ t)) = llength\ (\pi_c(ltake\ i\ t))$
using *the-enat-eSuc* **by** *simp*
with *nAct-def* **show** *?thesis* **by** *simp*
qed

lemma *nAct-not-active-same*:
assumes $enat\ n \leq (n'$::$enat)$
and $n'-1 < llength\ t$
and $\nexists k.\ enat\ k \geq n \wedge k < n' \wedge \|c\|_{lnth\ t\ k}$
shows $\langle c\ \#_{n'}\ t\rangle = \langle c\ \#_n\ t\rangle$
using *assms proj-not-active-same nAct-def* **by** *simp*

1.7.3 Active

lemma *nAct-active*[*simp*]:
fixes n::nat
and n'::nat
and t::$(cnf\ llist)$
and c::$'id$
assumes $enat\ i < llength\ t$
and $\|c\|_{lnth\ t\ i}$
shows $\langle c\ \#_{Suc\ i}\ t\rangle = eSuc\ (\langle c\ \#_i\ t\rangle)$
proof −
from *assms* **have** $\pi_c(ltake\ (Suc\ i)\ t) =$
$(\pi_c(ltake\ i\ t))\ @_l\ ((\sigma_c(lnth\ t\ i))\ \#_l\ []_l)$ **by** *simp*
hence $llength\ (\pi_c(ltake\ (enat\ (Suc\ i))\ t)) = eSuc\ (llength\ (\pi_c(ltake\ i\ t)))$
using *plus-1-eSuc one-eSuc* **by** *simp*
moreover have $llength\ (\pi_c(ltake\ i\ t)) \neq \infty$
using *llength-eq-infty-conv-lfinite*[*of* $\pi_c(ltake\ (enat\ i)\ t)$] **by** *simp*
ultimately have $llength\ (\pi_c(ltake\ (Suc\ i)\ t)) = eSuc\ (llength\ (\pi_c(ltake\ i\ t)))$
using *the-enat-eSuc* **by** *simp*
with *nAct-def* **show** *?thesis* **by** *simp*
qed

lemma *nAct-active-suc*:
fixes n::nat
and n'::$enat$
and t::$(cnf\ llist)$
and c::$'id$
assumes $\neg\ lfinite\ t \vee n'-1 < llength\ t$
and $n \leq i$
and $enat\ i < n'$
and $\|c\|_{lnth\ t\ i}$
and $\forall i'.\ (n \leq i' \wedge enat\ i' < n' \wedge i' < llength\ t \wedge \|c\|_{lnth\ t\ i'}) \longrightarrow (i' = i)$

shows $\langle c\ \#_{n'}\ t\rangle = eSuc\ (\langle c\ \#_n\ t\rangle)$
proof −
from *assms* **have** $\pi_c(ltake\ n'\ t) = (\pi_c(ltake\ (enat\ n)\ t))\ @_l\ ((\sigma_c(lnth\ t\ i))\ \#_l\ []_l)$
using *proj-active-append*[*of n i n' t c*] **by** *blast*
moreover have $llength\ ((\pi_c(ltake\ (enat\ n)\ t))\ @_l\ ((\sigma_c(lnth\ t\ i))\ \#_l\ []_l)) =$
$eSuc\ (llength\ (\pi_c(ltake\ (enat\ n)\ t)))$ **using** *one-eSuc eSuc-plus-1* **by** *simp*
ultimately show *?thesis* **using** *nAct-def* **by** *simp*
qed

lemma *nAct-less*:
assumes $enat\ k < llength\ t$
and $n \leq k$
and $k < (n'::enat)$
and $\|c\|_{lnth\ t\ k}$
shows $\langle c\ \#_n\ t\rangle < \langle c\ \#_{n'}\ t\rangle$
proof −
have $\langle c\ \#_k\ t\rangle \neq \infty$ **by** *simp*
then obtain *en* **where** *en-def*: $\langle c\ \#_k\ t\rangle = enat\ en$ **by** *blast*
moreover have $eSuc\ (enat\ en) \leq \langle c\ \#_{n'}\ t\rangle$
proof −
from *assms* **have** $Suc\ k \leq n'$ **using** *Suc-ile-eq* **by** *simp*
hence $\langle c\ \#_{Suc\ k}\ t\rangle \leq \langle c\ \#_{n'}\ t\rangle$ **using** *nAct-mono* **by** *simp*
moreover from *assms* **have** $\langle c\ \#_{Suc\ k}\ t\rangle = eSuc\ (\langle c\ \#_k\ t\rangle)$ **by** *simp*
ultimately have $eSuc\ (\langle c\ \#_k\ t\rangle) \leq \langle c\ \#_{n'}\ t\rangle$ **by** *simp*
thus *?thesis* **using** *en-def* **by** *simp*
qed
moreover have $enat\ en < eSuc\ (enat\ en)$ **by** *simp*
ultimately have $enat\ en < \langle c\ \#_{n'}\ t\rangle$ **using** *less-le-trans*[*of enat en eSuc* (*enat en*)] **by** *simp*
moreover have $\langle c\ \#_n\ t\rangle \leq enat\ en$
proof −
from *assms* **have** $\langle c\ \#_n\ t\rangle \leq \langle c\ \#_k\ t\rangle$ **using** *nAct-mono* **by** *simp*
thus *?thesis* **using** *en-def* **by** *simp*
qed
ultimately show *?thesis* **using** *le-less-trans*[*of* $\langle c\ \#_n\ t\rangle$] **by** *simp*
qed

lemma *nAct-less-active*:
assumes $n' - 1 < llength\ t$
and $\langle c\ \#_{enat\ n}\ t\rangle < \langle c\ \#_{n'}\ t\rangle$
shows $\exists\, i{\geq}n.\ i{<}n' \wedge \|c\|_{lnth\ t\ i}$
proof (*rule ccontr*)
assume $\neg\ (\exists\, i{\geq}n.\ i{<}n' \wedge \|c\|_{lnth\ t\ i})$
moreover have $enat\ n \leq n'$ **using** *assms*(2) *less-imp-le nAct-strict-mono-back* **by** *blast*
ultimately have $\langle c\ \#_n\ t\rangle = \langle c\ \#_{n'}\ t\rangle$ **using** ‹$n' - 1 < llength\ t$› *nAct-not-active-same* **by** *simp*
thus *False* **using** *assms* **by** *simp*
qed

1.7.4 Same and Not Same

lemma *nAct-same-not-active*:
assumes $\langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle = \langle c\ \#_n\ inf\text{-}llist\ t\rangle$
shows $\forall\, k{\geq}n.\ k{<}n' \longrightarrow \neg\ \|c\|_{t\ k}$
proof (*rule ccontr*)
assume $\neg(\forall\, k{\geq}n.\ k{<}n' \longrightarrow \neg\ \|c\|_{t\ k})$
then obtain k **where** $k{\geq}n$ **and** $k{<}n'$ **and** $\|c\|_{t\ k}$ **by** *blast*
hence $\langle c\ \#_{Suc\ k}\ inf\text{-}llist\ t\rangle = eSuc\ (\langle c\ \#_k\ inf\text{-}llist\ t\rangle)$ **by** *simp*

moreover have $\langle c\ \#_k\ inf\text{-}llist\ t\rangle \neq \infty$ **by** *simp*
ultimately have $\langle c\ \#_k\ inf\text{-}llist\ t\rangle < \langle c\ \#_{Suc\ k}\ inf\text{-}llist\ t\rangle$ **by** *fastforce*
moreover from $\langle n \leq k\rangle$ **have** $\langle c\ \#_n\ inf\text{-}llist\ t\rangle \leq \langle c\ \#_k\ inf\text{-}llist\ t\rangle$ **using** *nAct-mono* **by** *simp*
moreover from $\langle k < n'\rangle$ **have** $Suc\ k \leq n'$ **by** (*simp add*: *Suc-ile-eq*)
hence $\langle c\ \#_{Suc\ k}\ inf\text{-}llist\ t\rangle \leq \langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle$ **using** *nAct-mono* **by** *simp*
ultimately show *False* **using** *assms* **by** *simp*
qed

lemma *nAct-not-same-active*:
assumes $\langle c\ \#_{enat\ n}\ t\rangle < \langle c\ \#_{n'}\ t\rangle$
and $\neg$ *lfinite t* $\vee$ $n' - 1 <$ *llength t*
shows $\exists\,(i::nat) \geq n.$ *enat i* $<$ $n' \wedge i <$ *llength t* $\wedge\ \|c\|_{lnth\ t\ i}$
proof −
from *assms* **have** $llength(\pi_c(ltake\ n\ t)) < llength\ (\pi_c(ltake\ n'\ t))$ **using** *nAct-def* **by** *simp*
hence $\pi_c(ltake\ n'\ t) \neq \pi_c(ltake\ n\ t)$ **by** *auto*
moreover from *assms* **have** *enat n* $<$ n' **using** *nAct-strict-mono-back*[*of c enat n t n'*] **by** *simp*
ultimately show *?thesis* **using** *proj-not-same-active*[*of n n' t c*] *assms* **by** *simp*
qed

lemma *nAct-less-llength-active*:
assumes $x <$ *llength* $(\pi_c(t))$
and *enat* $x = \langle c\ \#_{enat\ n'}\ t\rangle$
shows $\exists\,(i::nat) \geq n'.$ $i <$ *llength t* $\wedge\ \|c\|_{lnth\ t\ i}$
proof −
have $llength(\pi_c(ltake\ n'\ t)) < llength\ (\pi_c(t))$ **using** *assms(1) assms(2) nAct-def* **by** *auto*
hence $llength(\pi_c(ltake\ n'\ t)) < llength\ (\pi_c(ltake\ (llength\ t)\ t))$ **by** (*simp add*: *ltake-all*)
hence $\langle c\ \#_{enat\ n'}\ t\rangle < \langle c\ \#_{llength\ t}\ t\rangle$ **using** *nAct-def* **by** *simp*
moreover have $\neg$ *lfinite t* $\vee$ *llength t* $-$ *1* $<$ *llength t*
proof (*rule Meson.imp-to-disjD*[*OF HOL.impI*])
assume *lfinite t*
hence *llength t* $\neq \infty$ **by** (*simp add*: *llength-eq-infty-conv-lfinite*)
moreover have *llength t*$>$*0*
proof −
from $\langle x <$ *llength* $(\pi_c(t))\rangle$ **have** *llength* $(\pi_c(t))>0$ **by** *auto*
thus *?thesis* **using** *proj-llength order.strict-trans2* **by** *blast*
qed
ultimately show *llength t* $-$ *1* $<$ *llength t* **by** (*metis One-nat-def* $\langle$*lfinite t*$\rangle$ *diff-Suc-less enat-ord-simps*(*2*) *idiff-enat-enat lfinite-conv-llength-enat one-enat-def zero-enat-def*)
qed
ultimately show *?thesis* **using** *nAct-not-same-active*[*of c n' t llength t*] **by** *simp*
qed

lemma *nAct-exists*:
assumes $x <$ *llength* $(\pi_c(t))$
shows $\exists\,(n'::nat).$ *enat* $x = \langle c\ \#_{n'}\ t\rangle$
proof −
have $x <$ *llength* $(\pi_c(t)) \longrightarrow (\exists\,(n'::nat).$ *enat* $x = \langle c\ \#_{n'}\ t\rangle)$
proof (*induction x*)
case *0*
thus *?case* **by** (*metis nAct-0 zero-enat-def*)
next
case (*Suc x*)
show *?case*
proof
assume $Suc\ x <$ *llength* $(\pi_c(t))$

hence $x < llength\ (\pi_c(t))$ **using** *Suc-ile-eq less-imp-le* **by** *auto*
with *Suc.IH* **obtain** n' **where** $enat\ x = \langle c\ \#_{enat\ n'}\ t\rangle$ **by** *blast*
with $\langle x < llength\ (\pi_c(t))\rangle$ **have** $\exists i{\geq}n'.\ i < llength\ t \wedge \|c\|_{lnth\ t\ i}$
using *nAct-less-llength-active*[*of x c t n'*] **by** *simp*
then obtain i **where** $i{\geq}n'$ **and** $i{<}llength\ t$ **and** $\|c\|_{lnth\ t\ i}$
and $\nexists k.\ n'{\leq}k \wedge k{<}i \wedge k{<}llength\ t \wedge \|c\|_{lnth\ t\ k}$ **using** *lActive-least*[*of n' t c*] **by** *auto*
moreover from $\langle i{<}llength\ t\rangle$ **have** $\neg\ lfinite\ t \vee enat\ (Suc\ i) - 1 < llength\ t$
by (*simp add*: *one-enat-def*)
moreover have $enat\ i < enat\ (Suc\ i)$ **by** *simp*
moreover have $\forall i'.\ (n' \leq i' \wedge enat\ i'{<}enat\ (Suc\ i) \wedge i'{<}llength\ t \wedge \|c\|_{lnth\ t\ i'}) \longrightarrow (i' = i)$
proof (*rule HOL.impI*[*THEN HOL.allI*])
fix i' **assume** $n' \leq i' \wedge enat\ i'{<}enat\ (Suc\ i) \wedge i'{<}llength\ t \wedge \|c\|_{lnth\ t\ i'}$
with $\langle\nexists k.\ n'{\leq}k \wedge k{<}i \wedge k{<}llength\ t \wedge \|c\|_{lnth\ t\ k}\rangle$ **show** $i'{=}i$ **by** *fastforce*
qed
ultimately have $\langle c\ \#_{Suc\ i}\ t\rangle = eSuc\ (\langle c\ \#_{n'}\ t\rangle)$ **using** *nAct-active-suc*[*of t Suc i n' i c*] **by** *simp*
with $\langle enat\ x = \langle c\ \#_{enat\ n'}\ t\rangle\rangle$ **have** $\langle c\ \#_{Suc\ i}\ t\rangle = eSuc\ (enat\ x)$ **by** *simp*
thus $\exists n'.\ enat\ (Suc\ x) = \langle c\ \#_{enat\ n'}\ t\rangle$ **by** (*metis eSuc-enat*)
qed
qed
with *assms* **show** *?thesis* **by** *simp*
qed

1.8 Projection and Activation

In the following we provide some properties about the relationship between the projection and activations operator.

lemma *nAct-le-proj*:
$\langle c\ \#_n\ t\rangle \leq llength\ (\pi_c(t))$
proof −
from *nAct-def* **have** $\langle c\ \#_n\ t\rangle = llength\ (\pi_c(ltake\ n\ t))$ **by** *simp*
moreover have $llength\ (\pi_c(ltake\ n\ t)) \leq llength\ (\pi_c(t))$
proof −
have *lprefix* $(ltake\ n\ t)\ t$ **by** *simp*
hence $lprefix\ (\pi_c(ltake\ n\ t))\ (\pi_c(t))$ **by** *simp*
hence $llength\ (\pi_c(ltake\ n\ t)) \leq llength\ (\pi_c(t))$ **using** *lprefix-llength-le* **by** *blast*
thus *?thesis* **by** *auto*
qed
thus *?thesis* **using** *nAct-def* **by** *simp*
qed

lemma *proj-nAct*:
assumes $enat\ n < llength\ t$
shows $\pi_c(ltake\ n\ t) = ltake\ (\langle c\ \#_n\ t\rangle)\ (\pi_c(t))$ (**is** *?lhs = ?rhs*)
proof −
have $?lhs = ltake\ (llength\ (\pi_c(ltake\ n\ t)))\ (\pi_c(ltake\ n\ t))$
using *ltake-all*[*of* $\pi_c(ltake\ n\ t)\ llength\ (\pi_c(ltake\ n\ t))$] **by** *simp*
also have $\ldots = ltake\ (llength\ (\pi_c(ltake\ n\ t)))\ ((\pi_c(ltake\ n\ t))\ @_l\ (\pi_c(ldrop\ n\ t)))$
using *ltake-lappend1*[*of* $llength\ (\pi_c(ltake\ (enat\ n)\ t))\ \pi_c(ltake\ n\ t)\ (\pi_c(ldrop\ n\ t))$] **by** *simp*
also have $\ldots = ltake\ (\langle c\ \#_n\ t\rangle)\ ((\pi_c(ltake\ n\ t))\ @_l\ (\pi_c(ldrop\ n\ t)))$ **using** *nAct-def* **by** *simp*
also have $\ldots = ltake\ (\langle c\ \#_n\ t\rangle)\ (\pi_c((ltake\ (enat\ n)\ t)\ @_l\ (ldrop\ n\ t)))$ **by** *simp*
also have $\ldots = ltake\ (\langle c\ \#_n\ t\rangle)\ (\pi_c(t))$ **using** *lappend-ltake-ldrop*[*of n t*] **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *proj-active-nth*:

assumes *enat* (*Suc i*) < *llength t* $\|c\|_{lnth\ t\ i}$
shows *lnth* ($\pi_c(t)$) (*the-enat* ($\langle c\ \#_i\ t\rangle$)) = σ_c(*lnth t i*)
proof −
from *assms* **have** *enat i* < *llength t* **using** *Suc-ile-eq*[*of i llength t*] **by** *auto*
with *assms* **have** π_c(*ltake* (*Suc i*) *t*) = (π_c(*ltake i t*)) $@_l$ ((σ_c(*lnth t i*)) $\#_l$ $[]_l$) **by** *simp*
moreover have *lnth* ((π_c(*ltake i t*)) $@_l$ ((σ_c(*lnth t i*)) $\#_l$ $[]_l$))
(*the-enat* (*llength* (π_c(*ltake i t*)))) = σ_c(*lnth t i*)
proof −
have ¬ *lnull* ((σ_c(*lnth t i*)) $\#_l$ $[]_l$) **by** *simp*
moreover have *lfinite* (π_c(*ltake i t*)) **by** *simp*
ultimately have *lnth* ((π_c(*ltake i t*)) $@_l$ ((σ_c(*lnth t i*)) $\#_l$ $[]_l$))
(*the-enat* (*llength* (π_c(*ltake i t*)))) = *lhd* ((σ_c(*lnth t i*)) $\#_l$ $[]_l$) **by** *simp*
also have ... = σ_c(*lnth t i*) **by** *simp*
finally show *lnth* ((π_c(*ltake i t*)) $@_l$ ((σ_c(*lnth t i*)) $\#_l$ $[]_l$))
(*the-enat* (*llength* (π_c(*ltake i t*)))) = σ_c(*lnth t i*) **by** *simp*
qed
ultimately have σ_c(*lnth t i*) = *lnth* (π_c(*ltake* (*Suc i*) *t*))
(*the-enat* (*llength* (π_c(*ltake i t*)))) **by** *simp*
also have ... = *lnth* (π_c(*ltake* (*Suc i*) *t*)) (*the-enat* ($\langle c\ \#_i\ t\rangle$)) **using** *nAct-def* **by** *simp*
also have ... = *lnth* (*ltake* ($\langle c\ \#_{Suc\ i}\ t\rangle$) ($\pi_c(t)$)) (*the-enat* ($\langle c\ \#_i\ t\rangle$))
using *proj-nAct*[*of Suc i t c*] *assms* **by** *simp*
also have ... = *lnth* ($\pi_c(t)$) (*the-enat* ($\langle c\ \#_i\ t\rangle$))
proof −
from *assms* **have** $\langle c\ \#_{Suc\ i}\ t\rangle$ = *eSuc* ($\langle c\ \#_i\ t\rangle$) **using** ‹*enat i* < *llength t*› **by** *simp*
moreover have $\langle c\ \#_i\ t\rangle$ < *eSuc* ($\langle c\ \#_i\ t\rangle$) **using** *iless-Suc-eq*[*of the-enat* ($\langle c\ \#_{enat\ i}\ t\rangle$)] **by** *simp*
ultimately have $\langle c\ \#_i\ t\rangle$ < ($\langle c\ \#_{Suc\ i}\ t\rangle$) **by** *simp*
hence *enat* (*the-enat* ($\langle c\ \#_{Suc\ i}\ t\rangle$)) > *enat* (*the-enat* ($\langle c\ \#_i\ t\rangle$)) **by** *simp*
thus *?thesis* **using** *lnth-ltake*[*of the-enat* ($\langle c\ \#_i\ t\rangle$) *the-enat* ($\langle c\ \#_{enat\ (Suc\ i)}\ t\rangle$) $\pi_c(t)$] **by** *simp*
qed
finally show *?thesis* **..**
qed

lemma *nAct-eq-proj*:
assumes ¬(∃ *i*≥*n*. $\|c\|_{lnth\ t\ i}$)
shows $\langle c\ \#_n\ t\rangle$ = *llength* ($\pi_c(t)$) (**is** *?lhs* = *?rhs*)
proof −
from *nAct-def* **have** *?lhs* = *llength* (π_c(*ltake n t*)) **by** *simp*
moreover from *assms* **have** ∀ (*n'*::*nat*)≤*llength t*. *n'*≥*n* ⟶ (¬ $\|c\|_{lnth\ t\ n'}$) **by** *simp*
hence $\pi_c(t)$ = π_c(*ltake n t*) **using** *proj-ltake* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *nAct-llength-proj*:
assumes ∃ *i*≥*n*. $\|c\|_{t\ i}$
shows *llength* (π_c(*inf-llist t*)) ≥ *eSuc* ($\langle c\ \#_n$ *inf-llist t*$\rangle$)
proof −
from ‹∃ *i*≥*n*. $\|c\|_{t\ i}$› **obtain** *i* **where** *i*≥*n* **and** $\|c\|_{t\ i}$
and ¬ (∃ *k*≥*n*. *k* < *i* ∧ *k* < *llength* (*inf-llist t*) ∧ $\|c\|_{t\ k}$)
using *lActive-least*[*of n inf-llist t c*] **by** *auto*
moreover have *llength* (π_c(*inf-llist t*)) ≥ $\langle c\ \#_{Suc\ i}$ *inf-llist t*$\rangle$ **using** *nAct-le-proj* **by** *simp*
moreover have *eSuc* ($\langle c\ \#_n$ *inf-llist t*$\rangle$) = $\langle c\ \#_{Suc\ i}$ *inf-llist t*$\rangle$
proof −
have *enat* (*Suc i*) < *llength* (*inf-llist t*) **by** *simp*
moreover have *i* < *Suc i* **by** *simp*
moreover from ‹¬ (∃ *k*≥*n*. *k* < *i* ∧ *k* < *llength* (*inf-llist t*) ∧ $\|c\|_{t\ k}$)›

have $\forall i'.\ n \leq i' \wedge i' < Suc\ i \wedge \|c\|_{lnth\ (inf\text{-}llist\ t)\ i'} \longrightarrow i' = i$ **by** *fastforce*
ultimately show *?thesis* **using** *nAct-active-suc* ‹$i \geq n$› ‹$\|c\|_{t\ i}$› **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed

1.9 Least not Active

In the following, we introduce an operator to obtain the least point in time before a certain point in time where a component was deactivated.

definition *lNAct* :: *'id* ⇒ (*nat* ⇒ *cnf*) ⇒ *nat* ⇒ *nat* (⟨- ⇐ -⟩-)
where $\langle c \Leftarrow t\rangle_n \equiv (LEAST\ n'.\ n{=}n' \vee (n'{<}n \wedge (\nexists k.\ k{\geq}n' \wedge k{<}n \wedge \|c\|_{t\ k})))$

lemma *lNact0*[*simp*]:
$\langle c \Leftarrow t\rangle_0 = 0$
by (*simp add*: *lNAct-def*)

lemma *lNact-least*:
assumes $n{=}n' \vee n'{<}n \wedge (\nexists k.\ k{\geq}n' \wedge k{<}n \wedge \|c\|_{t\ k})$
shows $\langle c \Leftarrow t\rangle_n \leq n'$
using *Least-le*[*of* $\lambda n'.\ n{=}n' \vee (n'{<}n \wedge (\nexists k.\ k{\geq}n' \wedge k{<}n \wedge \|c\|_{t\ k}))\ n'$] *lNAct-def* **using** *assms* **by** *auto*

lemma *lNAct-ex*: $\langle c \Leftarrow t\rangle_n{=}n \vee \langle c \Leftarrow t\rangle_n{<}n \wedge (\nexists k.\ k{\geq}\langle c \Leftarrow t\rangle_n \wedge k{<}n \wedge \|c\|_{t\ k})$
proof −
let *?P*=$\lambda n'.\ n{=}n' \vee n'{<}n \wedge (\nexists k.\ k{\geq}n' \wedge k{<}n \wedge \|c\|_{t\ k})$
from *lNAct-def* **have** $\langle c \Leftarrow t\rangle_n = (LEAST\ n'.\ ?P\ n')$ **by** *simp*
moreover have *?P n* **by** *simp*
with *LeastI* **have** *?P* $(LEAST\ n'.\ ?P\ n')$ **.**
ultimately show *?thesis* **by** *auto*
qed

lemma *lNact-notActive*:
fixes *c t n k*
assumes $k{\geq}\langle c \Leftarrow t\rangle_n$
and $k{<}n$
shows $\neg\|c\|_{t\ k}$
by (*metis assms lNAct-ex leD*)

lemma *lNactGe*:
fixes *c t n n'*
assumes $n' \geq \langle c \Leftarrow t\rangle_n$
and $\|c\|_{t\ n'}$
shows $n' \geq n$
using *assms lNact-notActive leI* **by** *blast*

lemma *lNactLe*[*simp*]:
fixes *n n'*
shows $\langle c \Leftarrow t\rangle_n \leq n$
using *lNAct-ex less-or-eq-imp-le* **by** *blast*

lemma *lNactLe-nact*:
fixes *n n'*
assumes $n'{=}n \vee (n'{<}n \wedge (\nexists k.\ k{\geq}n' \wedge k{<}n \wedge \|c\|_{t\ k}))$
shows $\langle c \Leftarrow t\rangle_n \leq n'$

using *assms lNAct-def Least-le*[*of* $\lambda n'.\ n{=}n' \vee (n'{<}n \wedge (\nexists k.\ k{\geq}n' \wedge k{<}n \wedge \|c\|_{t\ k}))$] **by** *auto*

lemma *lNact-active*:
fixes *cid t n*
assumes $\forall k{<}n.\ \|cid\|_{t\ k}$
shows $\langle cid \Leftarrow t\rangle_n = n$
using *assms lNAct-ex* **by** *blast*

lemma *nAct-mono-back*:
fixes *c t* **and** *n* **and** *n'*
assumes $\langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle \geq \langle c\ \#_n\ inf\text{-}llist\ t\rangle$
shows $n'{\geq}\langle c \Leftarrow t\rangle_n$
proof *cases*
assume $\langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle = \langle c\ \#_n\ inf\text{-}llist\ t\rangle$
thus *?thesis*
proof *cases*
assume $n'{\geq}n$
thus *?thesis*
by (*rule order-trans*[*OF lNactLe*])
next
assume $\neg\ n'{\geq}n$
hence $n'{<}n$ **by** *simp*
with $\langle\langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle = \langle c\ \#_n\ inf\text{-}llist\ t\rangle\rangle$ **have** $\nexists k.\ k{\geq}n' \wedge k{<}n \wedge \|c\|_{t\ k}$
by (*metis enat-ord-simps*(*1*) *enat-ord-simps*(*2*) *nAct-same-not-active*)
thus *?thesis* **using** *lNactLe-nact* **by** (*simp add*: $\langle n' < n\rangle$)
qed
next
assume $\neg\langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle = \langle c\ \#_n\ inf\text{-}llist\ t\rangle$
with *assms* **have** $\langle c\ \#_{enat\ n'}\ inf\text{-}llist\ t\rangle > \langle c\ \#_{enat\ n}\ inf\text{-}llist\ t\rangle$ **by** *simp*
hence $n' > n$ **using** *nAct-strict-mono-back*[*of c enat n inf-llist t enat n'*] **by** *simp*
thus *?thesis* **by** (*meson dual-order.strict-implies-order lNactLe le-trans*)
qed

lemma *nAct-mono-lNact*:
assumes $\langle c \Leftarrow t\rangle_n \leq n'$
shows $\langle c\ \#_n\ inf\text{-}llist\ t\rangle \leq \langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle$
proof −
have $\nexists k.\ k{\geq}\langle c \Leftarrow t\rangle_n \wedge k{<}n \wedge \|c\|_{t\ k}$ **using** *lNact-notActive* **by** *auto*
moreover have $enat\ n - 1 < llength\ (inf\text{-}llist\ t)$ **by** (*simp add*: *one-enat-def*)
moreover from $\langle\langle c \Leftarrow t\rangle_n \leq n'\rangle$ **have** $enat\ \langle c \Leftarrow t\rangle_n \leq enat\ n$ **by** *simp*
ultimately have $\langle c\ \#_n\ inf\text{-}llist\ t\rangle{=}\langle c\ \#_{\langle c \Leftarrow t\rangle_n}\ inf\text{-}llist\ t\rangle$ **using** *nAct-not-active-same* **by** *simp*
thus *?thesis* **using** *nAct-mono assms* **by** *simp*
qed

1.10 Next Active

In the following, we introduce an operator to obtain the next point in time when a component is activated.

definition *nxtAct* :: *'id* ⇒ (*nat* ⇒ *cnf*) ⇒ *nat* ⇒ *nat* ($\langle$- → -$\rangle$_)
where $\langle c \rightarrow t\rangle_n \equiv (THE\ n'.\ n'{\geq}n \wedge \|c\|_{t\ n'} \wedge (\nexists k.\ k{\geq}n \wedge k{<}n' \wedge \|c\|_{t\ k}))$

lemma *nxtActI*:
fixes *n*::*nat*
and *t*::*nat* ⇒ *cnf*
and *c*::*'id*

assumes $\exists\, i{\geq}n.\ \|c\|_{t\ i}$
shows $\langle c \to t\rangle_n \geq n \wedge \|c\|_{t\ \langle c \to t\rangle_n} \wedge (\nexists k.\ k{\geq}n \wedge k{<}\langle c \to t\rangle_n \wedge \|c\|_{t\ k})$
proof −
let *?P* = *THE* $n'.\ n'{\geq}n \wedge \|c\|_{t\ n'} \wedge (\nexists k.\ k{\geq}n \wedge k{<}n' \wedge \|c\|_{t\ k})$
from *assms* **obtain** i **where** $i{\geq}n \wedge \|c\|_{t\ i} \wedge (\nexists k.\ k{\geq}n \wedge k{<}i \wedge \|c\|_{t\ k})$
using *lActive-least*[*of n inf-llist t c*] **by** *auto*
moreover have $(\bigwedge x.\ n \leq x \wedge \|c\|_{t\ x} \wedge \neg\ (\exists\, k{\geq}n.\ k < x \wedge \|c\|_{t\ k}) \Longrightarrow x = i)$
proof −
fix x **assume** $n \leq x \wedge \|c\|_{t\ x} \wedge \neg\ (\exists\, k{\geq}n.\ k < x \wedge \|c\|_{t\ k})$
show $x = i$
proof (*rule ccontr*)
assume $\neg\ (x = i)$
thus *False* **using** ‹$i{\geq}n \wedge \|c\|_{t\ i} \wedge (\nexists k.\ k{\geq}n \wedge k{<}i \wedge \|c\|_{t\ k})$›
‹$n \leq x \wedge \|c\|_{t\ x} \wedge \neg\ (\exists\, k{\geq}n.\ k < x \wedge \|c\|_{t\ k})$› **by** *fastforce*
qed
qed
ultimately have $(?P) \geq n \wedge \|c\|_{t\ (?P)} \wedge (\nexists k.\ k{\geq}n \wedge k{<}?P \wedge \|c\|_{t\ k})$
using *theI*[*of* $\lambda n'.\ n'{\geq}n \wedge \|c\|_{t\ n'} \wedge (\nexists k.\ k{\geq}n \wedge k{<}n' \wedge \|c\|_{t\ k})$] **by** *blast*
thus *?thesis* **using** *nxtAct-def*[*of c t n*] **by** *metis*
qed

lemma *nxtActLe*:
fixes $n\ n'$
assumes $\exists\, i{\geq}n.\ \|c\|_{t\ i}$
shows $n \leq \langle c \to t\rangle_n$
by (*simp add*: *assms nxtActI*)

lemma *nxtAct-eq*:
assumes $n'{\geq}n$
and $\|c\|_{t\ n'}$
and $\forall\, n''{\geq}n.\ n''{<}n' \longrightarrow \neg\ \|c\|_{t\ n''}$
shows $n' = \langle c \to t\rangle_n$
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *nxtActI linorder-neqE-nat nxtActLe*)

lemma *nxtAct-active*:
fixes i::*nat*
and t::*nat* ⇒ *cnf*
and c::$'id$
assumes $\|c\|_{t\ i}$
shows $\langle c \to t\rangle_i = i$ **by** (*metis assms le-eq-less-or-eq nxtActI*)

lemma *nxtActive-no-active*:
assumes $\exists!\, i.\ i{\geq}n \wedge \|c\|_{t\ i}$
shows $\neg\ (\exists\, i'{\geq}Suc\ \langle c \to t\rangle_n.\ \|c\|_{t\ i'})$
proof
assume $\exists\, i'{\geq}Suc\ \langle c \to t\rangle_n.\ \|c\|_{t\ i'}$
then obtain i' **where** $i'{\geq}Suc\ \langle c \to t\rangle_n$ **and** $\|c\|_{t\ i'}$ **by** *auto*
moreover from *assms*(*1*) **have** $\langle c \to t\rangle_n{\geq}n$ **using** *nxtActI* **by** *auto*
ultimately have $i'{\geq}n$ **and** $\|c\|_{t\ i'}$ **and** $i'{\neq}\langle c \to t\rangle_n$ **by** *auto*
moreover from *assms*(*1*) **have** $\|c\|_{t\ \langle c \to t\rangle_n}$ **and** $\langle c \to t\rangle_n{\geq}n$ **using** *nxtActI* **by** *auto*
ultimately show *False* **using** *assms*(*1*) **by** *auto*
qed

lemma *nxt-geq-lNact*[*simp*]:
assumes $\exists\, i{\geq}n.\ \|c\|_{t\ i}$

shows $\langle c \rightarrow t\rangle_n \geq \langle c \Leftarrow t\rangle_n$
proof −
from *assms* **have** $n \leq \langle c \rightarrow t\rangle_n$ **using** *nxtActLe* **by** *simp*
moreover have $\langle c \Leftarrow t\rangle_n \leq n$ **by** *simp*
ultimately show *?thesis* **by** *arith*
qed

lemma *active-geq-nxtAct*:
assumes $\|c\|_{t\ i}$
and *the-enat* $(\langle c\ \#_i$ *inf-llist* $t\rangle) \geq$ *the-enat* $(\langle c\ \#_n$ *inf-llist* $t\rangle)$
shows $i \geq \langle c \rightarrow t\rangle_n$
proof *cases*
assume $\langle c\ \#_i$ *inf-llist* $t\rangle = \langle c\ \#_n$ *inf-llist* $t\rangle$
show *?thesis*
proof (*rule ccontr*)
assume $\neg\ i \geq \langle c \rightarrow t\rangle_n$
hence $i < \langle c \rightarrow t\rangle_n$ **by** *simp*
with $\langle\langle c\ \#_i$ *inf-llist* $t\rangle = \langle c\ \#_n$ *inf-llist* $t\rangle\rangle$ **have** $\neg\ (\exists\, k \geq i.\ k < n \wedge \|c\|_{t\ k})$
by (*metis enat-ord-simps(1) leD leI nAct-same-not-active*)
moreover have $\neg\ (\exists\, k \geq n.\ k < \langle c \rightarrow t\rangle_n \wedge \|c\|_{t\ k})$ **using** *nxtActI* **by** *blast*
ultimately have $\neg\ (\exists\, k \geq i.\ k < \langle c \rightarrow t\rangle_n \wedge \|c\|_{t\ k})$ **by** *auto*
with $\langle i < \langle c \rightarrow t\rangle_n\rangle$ **show** *False* **using** $\langle\|c\|_{t\ i}\rangle$ **by** *simp*
qed
next
assume $\neg\langle c\ \#_i$ *inf-llist* $t\rangle = \langle c\ \#_n$ *inf-llist* $t\rangle$
moreover from $\langle$*the-enat* $(\langle c\ \#_i$ *inf-llist* $t\rangle) \geq$ *the-enat* $(\langle c\ \#_n$ *inf-llist* $t\rangle)\rangle$
have $\langle c\ \#_i$ *inf-llist* $t\rangle \geq \langle c\ \#_n$ *inf-llist* $t\rangle$
by (*metis enat.distinct(2) enat-ord-simps(1) nAct-enat-the-nat*)
ultimately have $\langle c\ \#_i$ *inf-llist* $t\rangle > \langle c\ \#_n$ *inf-llist* $t\rangle$ **by** *simp*
hence $i > n$ **using** *nAct-strict-mono-back*[*of c n inf-llist t i*] **by** *simp*
with $\langle\|c\|_{t\ i}\rangle$ **show** *?thesis* **by** (*meson dual-order.strict-implies-order leI nxtActI*)
qed

lemma *nAct-same*:
assumes $\langle c \Leftarrow t\rangle_n \leq n'$ **and** $n' \leq \langle c \rightarrow t\rangle_n$
shows *the-enat* $(\langle c\ \#_{enat\ n'}$ *inf-llist* $t\rangle) =$ *the-enat* $(\langle c\ \#_{enat\ n}$ *inf-llist* $t\rangle)$
proof *cases*
assume $n \leq n'$
moreover have $n' - 1 <$ *llength* (*inf-llist t*) **by** *simp*
moreover have $\neg\ (\exists\, i \geq n.\ i < n' \wedge \|c\|_{t\ i})$ **by** (*meson assms(2) less-le-trans nxtActI*)
ultimately show *?thesis* **using** *nAct-not-active-same* **by** (*simp add*: *one-enat-def*)
next
assume $\neg\ n \leq n'$
hence $n' < n$ **by** *simp*
moreover have $n - 1 <$ *llength* (*inf-llist t*) **by** *simp*
moreover have $\neg\ (\exists\, i \geq n'.\ i < n \wedge \|c\|_{t\ i})$ **by** (*metis* $\langle\neg\ n \leq n'\rangle$ *assms(1) dual-order.trans lNAct-ex*)
ultimately show *?thesis* **using** *nAct-not-active-same*[*of n' n*] **by** (*simp add*: *one-enat-def*)
qed

lemma *nAct-mono-nxtAct*:
assumes $\exists\, i \geq n.\ \|c\|_{t\ i}$
and $\langle c \rightarrow t\rangle_n \leq n'$
shows $\langle c\ \#_n$ *inf-llist* $t\rangle \leq \langle c\ \#_{n'}$ *inf-llist* $t\rangle$
proof −
from *assms* **have** $\langle c\ \#_{\langle c \rightarrow t\rangle_n}$ *inf-llist* $t\rangle \leq \langle c\ \#_{n'}$ *inf-llist* $t\rangle$ **using** *nAct-mono assms* **by** *simp*

moreover have $\langle c \#_{\langle c \to t\rangle_n}$ *inf-llist* $t\rangle = \langle c \#_n$ *inf-llist* $t\rangle$
proof −
from *assms* **have** $\nexists\, k.\ k \geq n \wedge k < \langle c \to t\rangle_n \wedge \|c\|_{t\ k}$ **and** $n \leq \langle c \to t\rangle_n$ **using** *nxtActI* **by** *auto*
moreover have *enat* $\langle c \to t\rangle_n - 1 <$ *llength* (*inf-llist* t) **by** (*simp add*: *one-enat-def*)
ultimately show *?thesis* **using** *nAct-not-active-same*[*of* n $\langle c \to t\rangle_n$] **by** *auto*
qed
ultimately show *?thesis* **by** *simp*
qed

1.11 Latest Activation

In the following, we introduce an operator to obtain the latest point in time when a component is activated.

abbreviation *latestAct-cond*:: *'id* ⇒ *trace* ⇒ *nat* ⇒ *nat* ⇒ *bool*
where *latestAct-cond* $c\ t\ n\ n' \equiv n' < n \wedge \|c\|_{t\ n'}$

definition *latestAct*:: *'id* ⇒ *trace* ⇒ *nat* ⇒ *nat* ($\langle$- ← -$\rangle$_)
where *latestAct* $c\ t\ n =$ (*GREATEST* n'. *latestAct-cond* $c\ t\ n\ n'$)

lemma *latestActEx*:
assumes $\exists\, n' < n.\ \|nid\|_{t\ n'}$
shows $\exists\, n'$. *latestAct-cond nid t n* $n' \wedge (\forall\, n''$. *latestAct-cond nid t n* $n'' \longrightarrow n'' \leq n')$
proof −
from *assms* **obtain** n' **where** *latestAct-cond nid t n* n' **by** *auto*
moreover have $\forall\, n'' > n.\ \neg$ *latestAct-cond nid t n* n'' **by** *simp*
ultimately obtain n' **where** *latestAct-cond nid t n* $n' \wedge (\forall\, n''$. *latestAct-cond nid t n* $n'' \longrightarrow n'' \leq n')$
using *boundedGreatest*[*of latestAct-cond nid t n* n'] **by** *blast*
thus *?thesis* **..**
qed

lemma *latestAct-prop*:
assumes $\exists\, n' < n.\ \|nid\|_{t\ n'}$
shows $\|nid\|_{t\ (latestAct\ nid\ t\ n)}$ **and** *latestAct nid t* $n < n$
proof −
from *assms latestActEx* **have** *latestAct-cond nid t n* (*GREATEST* x. *latestAct-cond nid t n* x)
using *GreatestI-ex-nat*[*of latestAct-cond nid t n*] **by** *blast*
thus $\|nid\|_{t\ \langle nid \leftarrow t\rangle_n}$ **and** *latestAct nid t* $n < n$ **using** *latestAct-def* **by** *auto*
qed

lemma *latestAct-less*:
assumes *latestAct-cond nid t n* n'
shows $n' \leq \langle nid \leftarrow t\rangle_n$
proof −
from *assms latestActEx* **have** $n' \leq$ (*GREATEST* x. *latestAct-cond nid t n* x)
using *Greatest-le-nat*[*of latestAct-cond nid t n*] **by** *blast*
thus *?thesis* **using** *latestAct-def* **by** *auto*
qed

lemma *latestActNxt*:
assumes $\exists\, n' < n.\ \|nid\|_{t\ n'}$
shows $\langle nid \to t\rangle_{\langle nid \leftarrow t\rangle_n} = \langle nid \leftarrow t\rangle_n$
using *assms latestAct-prop*(*1*) *nxtAct-active* **by** *auto*

lemma *latestActNxtAct*:
assumes $\exists\, n'{\geq}n.\ \|tid\|_{t\ n'}$
and $\exists\, n'{<}n.\ \|tid\|_{t\ n'}$
shows $\langle tid \rightarrow t\rangle_n > \langle tid \leftarrow t\rangle_n$
by (*meson assms latestAct-prop*(*2*) *less-le-trans nxtActI zero-le*)

lemma *latestActless*:
assumes $\exists\, n'{\geq}n_s.\ n'{<}n \wedge \|nid\|_{t\ n'}$
shows $\langle nid \leftarrow t\rangle_n{\geq}n_s$
by (*meson assms dual-order.trans latestAct-less*)

lemma *latestActEq*:
fixes *nid*::*'id*
assumes $\|nid\|_{t\ n'}$ **and** $\neg(\exists\, n''{>}n'.\ n''{<}n \wedge \|nid\|_{t\ n'})$ **and** $n'{<}n$
shows $\langle nid \leftarrow t\rangle_n = n'$
using *latestAct-def*
proof
have (*GREATEST n'. latestAct-cond nid t n n'*) $= n'$
proof (*rule Greatest-equality*[*of latestAct-cond nid t n n'*])
from *assms*(*1*) *assms* (*3*) **show** *latestAct-cond nid t n n'* **by** *simp*
next
fix *y* **assume** *latestAct-cond nid t n y*
hence $\|nid\|_{t\ y}$ **and** $y{<}n$ **by** *auto*
thus $y \leq n'$ **using** *assms*(*1*) *assms* (*2*) *leI* **by** *blast*
qed
thus $n' =$ (*GREATEST n'. latestAct-cond nid t n n'*) **by** *simp*
qed

1.12 Last Activation

In the following we introduce an operator to obtain the latest point in time where a certain component was activated within a certain configuration trace.

definition *lActive* :: *'id* ⇒ (*nat* ⇒ *cnf*) ⇒ *nat* (⟨- ∧ -⟩)
where $\langle c \wedge t\rangle \equiv$ (*GREATEST i*. $\|c\|_{t\ i}$)

lemma *lActive-active*:
assumes $\|c\|_{t\ i}$
and $\forall\, n' > n.\ \neg\ (\|c\|_{t\ n'})$
shows $\|c\|_{t\ (\langle c \wedge t\rangle)}$
proof −
from *assms* **obtain** i' **where** $\|c\|_{t\ i'}$ **and** $\bigwedge y.\ \|c\|_{t\ y} \Longrightarrow y \leq i'$
using *boundedGreatest*[*of* $\lambda i'.\ \|c\|_{t\ i'}$ *i n*] **by** *blast*
thus *?thesis* **using** *lActive-def Nat.GreatestI-nat*[*of* $\lambda i'.\ \|c\|_{t\ i'}$] **by** *metis*
qed

lemma *lActive-less*:
assumes $\|c\|_{t\ i}$
and $\forall\, n' > n.\ \neg\ (\|c\|_{t\ n'})$
shows $\langle c \wedge t\rangle \leq n$
proof (*rule ccontr*)
assume $\neg\ \langle c \wedge t\rangle \leq n$
hence $\langle c \wedge t\rangle > n$ **by** *simp*
moreover from *assms* **have** $\|c\|_{t\ (\langle c \wedge t\rangle)}$ **using** *lActive-active* **by** *simp*
ultimately show *False* **using** *assms* **by** *simp*
qed

lemma *lActive-greatest*:
assumes $\|c\|_{t\ i}$
and $\forall n' > n.\ \neg\ (\|c\|_{t\ n'})$
shows $i \leq \langle c \wedge t \rangle$
proof −
from *assms* **obtain** i' **where** $\|c\|_{t\ i'}$ **and** $\bigwedge y.\ \|c\|_{t\ y} \Longrightarrow y \leq i'$
using *boundedGreatest*[*of* $\lambda i'.\ \|c\|_{t\ i'}\ i\ n$] **by** *blast*
with *assms* **show** *?thesis* **using** *lActive-def Nat.Greatest-le-nat*[*of* $\lambda i'.\ \|c\|_{t\ i'}\ i$] **by** *metis*
qed

lemma *lActive-greater-active*:
assumes $n > \langle c \wedge t \rangle$
and $\forall n'' > n'.\ \neg\ \|c\|_{t\ n''}$
shows $\neg\ \|c\|_{t\ n}$
proof (*rule ccontr*)
assume $\neg\ \neg\ \|c\|_{t\ n}$
with ‹$\forall n'' > n'.\ \neg\ \|c\|_{t\ n''}$› **have** $n \leq \langle c \wedge t \rangle$ **using** *lActive-greatest* **by** *simp*
thus *False* **using** *assms* **by** *simp*
qed

lemma *lActive-greater-active-all*:
assumes $\forall n'' > n'.\ \neg\ \|c\|_{t\ n''}$
shows $\neg(\exists n > \langle c \wedge t \rangle.\ \|c\|_{t\ n})$
proof (*rule ccontr*)
assume $\neg\neg(\exists n > \langle c \wedge t \rangle.\ \|c\|_{t\ n})$
then obtain n **where** $n{>}\langle c \wedge t \rangle$ **and** $\|c\|_{t\ n}$ **by** *blast*
with ‹$\forall n'' > n'.\ \neg\ (\|c\|_{t\ n''})$› **have** $\neg\ \|c\|_{t\ n}$ **using** *lActive-greater-active* **by** *simp*
with ‹$\|c\|_{t\ n}$› **show** *False* **by** *simp*
qed

lemma *lActive-equality*:
assumes $\|c\|_{t\ i}$
and $(\bigwedge x.\ \|c\|_{t\ x} \Longrightarrow x \leq i)$
shows $\langle c \wedge t \rangle = i$ **unfolding** *lActive-def* **using** *assms Greatest-equality*[*of* $\lambda i'.\ \|c\|_{t\ i'}$] **by** *simp*

lemma *nxtActive-lactive*:
assumes $\exists i{\geq}n.\ \|c\|_{t\ i}$
and $\neg\ (\exists i{>}\langle c \rightarrow t\rangle_n.\ \|c\|_{t\ i})$
shows $\langle c \rightarrow t\rangle_n{=}\langle c \wedge t\rangle$
proof −
from *assms*(*1*) **have** $\|c\|_{t\ \langle c \rightarrow t\rangle_n}$ **using** *nxtActI* **by** *auto*
moreover from *assms* **have** $\neg\ (\exists i'{\geq}Suc\ \langle c \rightarrow t\rangle_n.\ \|c\|_{t\ i'})$ **using** *nxtActive-no-active* **by** *simp*
hence $(\bigwedge x.\ \|c\|_{t\ x} \Longrightarrow x \leq \langle c \rightarrow t\rangle_n)$ **using** *not-less-eq-eq* **by** *auto*
ultimately show *?thesis* **using** ‹$\neg\ (\exists i'{\geq}Suc\ \langle c \rightarrow t\rangle_n.\ \|c\|_{t\ i'})$› *lActive-equality* **by** *simp*
qed

1.13 Mapping Time Points

In the following we introduce two operators to map time-points between configuration traces and behavior traces.

1.13.1 Configuration Trace to Behavior Trace

First we provide an operator which maps a point in time of a configuration trace to the corresponding point in time of a behavior trace.

definition *cnf2bhv* :: *'id* ⇒ (*nat* ⇒ *cnf*) ⇒ *nat* ⇒ *nat* (_↓_(-) [*150*,*150*,*150*] *110*)
where $_c\!\downarrow_t(n) \equiv$ *the-enat*(*llength* (π_c(*inf-llist t*))) − *1* + (*n* − ⟨*c* ∧ *t*⟩)

lemma *cnf2bhv-mono*:
assumes *n'*≥*n*
shows $_c\!\downarrow_t(n') \geq {}_c\!\downarrow_t(n)$
by (*simp add*: *assms cnf2bhv-def diff-le-mono*)

lemma *cnf2bhv-mono-strict*:
assumes *n*≥⟨*c* ∧ *t*⟩ **and** *n'*>*n*
shows $_c\!\downarrow_t(n') > {}_c\!\downarrow_t(n)$
using *assms cnf2bhv-def* **by** *auto*

Note that the functions are nat, that means that also in the case the difference is negative they will return a 0!

lemma *cnf2bhv-ge-llength*[*simp*]:
assumes *n*≥⟨*c* ∧ *t*⟩
shows $_c\!\downarrow_t(n) \geq$ *the-enat*(*llength* (π_c(*inf-llist t*))) − *1*
using *assms cnf2bhv-def* **by** *simp*

lemma *cnf2bhv-greater-llength*[*simp*]:
assumes *n*>⟨*c* ∧ *t*⟩
shows $_c\!\downarrow_t(n) >$ *the-enat*(*llength* (π_c(*inf-llist t*))) − *1*
using *assms cnf2bhv-def* **by** *simp*

lemma *cnf2bhv-suc*[*simp*]:
assumes *n*≥⟨*c* ∧ *t*⟩
shows $_c\!\downarrow_t$(*Suc n*) = *Suc* ($_c\!\downarrow_t(n)$)
using *assms cnf2bhv-def* **by** *simp*

lemma *cnf2bhv-lActive*[*simp*]:
shows $_c\!\downarrow_t$(⟨*c* ∧ *t*⟩) = *the-enat*(*llength* (π_c(*inf-llist t*))) − *1*
using *cnf2bhv-def* **by** *simp*

lemma *cnf2bhv-lnth-lappend*:
assumes *act*: ∃ *i*. $\|c\|_{t\ i}$
and *nAct*: ∄ *i*. *i*≥*n* ∧ $\|c\|_{t\ i}$
shows *lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*)) ($_c\!\downarrow_t(n)$) = *lnth* (*inf-llist t'*) (*n* − ⟨*c* ∧ *t*⟩ − *1*)
(**is** *?lhs* = *?rhs*)
proof −
from *nAct* **have** *lfinite* (π_c(*inf-llist t*)) **using** *proj-finite2* **by** *auto*
then obtain *k* **where** *k-def*: *llength* (π_c(*inf-llist t*)) = *enat k* **using** *lfinite-llength-enat* **by** *blast*
moreover have *k* ≤ $_c\!\downarrow_t(n)$
proof −
from *nAct* **have** ∄ *i*. *i*>*n*−*1* ∧ $\|c\|_{t\ i}$ **by** *simp*
with *act* **have** ⟨*c* ∧ *t*⟩ ≤ *n*−*1* **using** *lActive-less* **by** *auto*
moreover have *n*>*0* **using** *act nAct* **by** *auto*
ultimately have ⟨*c* ∧ *t*⟩ < *n* **by** *simp*
hence *the-enat* (*llength* (π_c*inf-llist t*)) − *1* < $_c\!\downarrow_t(n)$ **using** *cnf2bhv-greater-llength* **by** *simp*
with *k-def* **show** *?thesis* **by** *simp*
qed

ultimately have *?lhs = lnth (inf-llist t′) (c↓t(n) − k)* **using** *lnth-lappend2* **by** *blast*
moreover have $_{c\downarrow t}(n) - k = n - \langle c \wedge t\rangle - 1$
proof −
from *cnf2bhv-def* **have** $_{c\downarrow t}(n) - k = \textit{the-enat}\ (\textit{llength}\ (\pi_c \textit{inf-llist}\ t)) - 1 + (n - \langle c \wedge t\rangle) - k$
by *simp*
also have $\ldots = \textit{the-enat}\ (\textit{llength}\ (\pi_c \textit{inf-llist}\ t)) - 1 + (n - \langle c \wedge t\rangle) -$
the-enat (llength (πc(inf-llist t))) **using** *k-def* **by** *simp*
also have $\ldots = \textit{the-enat}\ (\textit{llength}\ (\pi_c \textit{inf-llist}\ t)) + (n - \langle c \wedge t\rangle) - 1 -$
the-enat (llength (πc(inf-llist t)))
proof −
have $\exists\, i.\ \textit{enat}\ i < \textit{llength}\ (\textit{inf-llist}\ t) \wedge \|c\|_{\textit{lnth}\ (\textit{inf-llist}\ t)\ i}$ **by** (*simp add*: *act*)
hence $\textit{llength}\ (\pi_c \textit{inf-llist}\ t) \geq 1$ **using** *proj-one* **by** *simp*
moreover from *k-def* **have** $\textit{llength}\ (\pi_c \textit{inf-llist}\ t) \neq \infty$ **by** *simp*
ultimately have $\textit{the-enat}\ (\textit{llength}\ (\pi_c \textit{inf-llist}\ t)) \geq 1$ **by** (*simp add*: *k-def one-enat-def*)
thus *?thesis* **by** *simp*
qed
also have $\ldots = \textit{the-enat}\ (\textit{llength}\ (\pi_c \textit{inf-llist}\ t)) + (n - \langle c \wedge t\rangle) -$
the-enat (llength (πc(inf-llist t))) − *1* **by** *simp*
also have $\ldots = n - \langle c \wedge t\rangle - 1$ **by** *simp*
finally show *?thesis* **.**
qed
ultimately show *?thesis* **by** *simp*
qed

lemma *nAct-cnf2proj-Suc-dist*:
assumes $\exists\, i{\geq}n.\ \|c\|_{t\ i}$
and $\neg(\exists\, i{>}\langle c \rightarrow t\rangle_n.\ \|c\|_{t\ i})$
shows $\textit{Suc}\ (\textit{the-enat}\ \langle c\ \#_{\textit{enat}\ n} \textit{inf-llist}\ t\rangle){=}_{c\downarrow t}(\textit{Suc}\ \langle c \rightarrow t\rangle_n)$
proof −
have $\textit{the-enat}\ \langle c\ \#_{\textit{enat}\ n} \textit{inf-llist}\ t\rangle = {}_{c\downarrow t}(\langle c \rightarrow t\rangle_n)$ (**is** *?LHS = ?RHS*)
proof −
from *assms* **have** $\textit{?RHS} = \textit{the-enat}(\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1$
using *nxtActive-lactive*[*of n c t*] **by** *simp*
also have $\textit{llength}\ (\pi_c(\textit{inf-llist}\ t)) = \textit{eSuc}\ (\langle c\ \#_{\langle c \rightarrow t\rangle_n}\ \textit{inf-llist}\ t\rangle)$
proof −
from *assms* **have** $\neg\ (\exists\, i'{\geq}\ \textit{Suc}\ (\langle c \rightarrow t\rangle_n).\ \|c\|_{t\ i'})$ **using** *nxtActive-no-active* **by** *simp*
hence $\langle c\ \#_{\textit{Suc}\ (\langle c \rightarrow t\rangle_n)}\ \textit{inf-llist}\ t\rangle = \textit{llength}\ (\pi_c(\textit{inf-llist}\ t))$
using *nAct-eq-proj*[*of Suc (⟨c → t⟩n) c inf-llist t*] **by** *simp*
moreover from *assms*(*1*) **have** $\|c\|_{t\ (\langle c \rightarrow t\rangle_n)}$ **using** *nxtActI* **by** *blast*
hence $\langle c\ \#_{\textit{Suc}\ (\langle c \rightarrow t\rangle_n)}\ \textit{inf-llist}\ t\rangle = \textit{eSuc}\ (\langle c\ \#_{\langle c \rightarrow t\rangle_n}\ \textit{inf-llist}\ t\rangle)$ **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
also have $\textit{the-enat}(\textit{eSuc}\ (\langle c\ \#_{\langle c \rightarrow t\rangle_n}\ \textit{inf-llist}\ t\rangle)) - 1 = (\langle c\ \#_{\langle c \rightarrow t\rangle_n}\ \textit{inf-llist}\ t\rangle)$
proof −
have $\langle c\ \#_{\langle c \rightarrow t\rangle_n}\ \textit{inf-llist}\ t\rangle \neq \infty$ **by** *simp*
hence $\textit{the-enat}(\textit{eSuc}\ (\langle c\ \#_{\langle c \rightarrow t\rangle_n}\ \textit{inf-llist}\ t\rangle)) = \textit{Suc}(\textit{the-enat}(\langle c\ \#_{\langle c \rightarrow t\rangle_n}\ \textit{inf-llist}\ t\rangle))$
using *the-enat-eSuc* **by** *simp*
thus *?thesis* **by** *simp*
qed
also have $\ldots = \textit{?LHS}$
proof −
have $\textit{enat}\ \langle c \rightarrow t\rangle_n - 1 < \textit{llength}\ (\textit{inf-llist}\ t)$ **by** (*simp add*: *one-enat-def*)
moreover from *assms*(*1*) **have** $\langle c \rightarrow t\rangle_n{\geq}n$ **and**
$\nexists\, k.\ \textit{enat}\ n \leq \textit{enat}\ k \wedge \textit{enat}\ k < \textit{enat}\ \langle c \rightarrow t\rangle_n \wedge \|c\|_{\textit{lnth}\ (\textit{inf-llist}\ t)\ k}$ **using** *nxtActI* **by** *auto*

ultimately have $\langle c \#_{enat\ \langle c \to t\rangle_n} inf\text{-}llist\ t\rangle = \langle c \#_{enat\ n} inf\text{-}llist\ t\rangle$
using *nAct-not-active-same*[*of* $n\ \langle c \to t\rangle_n$ *inf-llist t c*] **by** *simp*
moreover have $\langle c \#_{enat\ n} inf\text{-}llist\ t\rangle \neq \infty$ **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
finally show *?thesis* **by** *fastforce*
qed
moreover from *assms* **have** $\langle c \to t\rangle_n = \langle c \wedge t\rangle$ **using** *nxtActive-lactive* **by** *simp*
hence $Suc\ ({}_c\downarrow_t(\langle c \to t\rangle_n)) = {}_c\downarrow_t(Suc\ \langle c \to t\rangle_n)$ **using** *cnf2bhv-suc*[**where** $n = \langle c \to t\rangle_n$] **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

1.13.2 Behavior Trace to Configuration Trace

Next we define an operator to map a point in time of a behavior trace back to a corresponding point in time for a configuration trace.

definition *bhv2cnf* :: *'id* ⇒ (*nat* ⇒ *cnf*) ⇒ *nat* ⇒ *nat* (-↑-(-) [*150*,*150*,*150*] *110*)
where ${}_c\uparrow_t(n) \equiv \langle c \wedge t\rangle + (n - (the\text{-}enat(llength\ (\pi_c(inf\text{-}llist\ t))) - 1))$

lemma *bhv2cnf-mono*:
assumes $n' \geq n$
shows ${}_c\uparrow_t(n') \geq {}_c\uparrow_t(n)$
by (*simp add*: *assms bhv2cnf-def diff-le-mono*)

lemma *bhv2cnf-mono-strict*:
assumes $n' > n$
and $n \geq the\text{-}enat\ (llength\ (\pi_c(inf\text{-}llist\ t))) - 1$
shows ${}_c\uparrow_t(n') > {}_c\uparrow_t(n)$
using *assms bhv2cnf-def* **by** *auto*

Note that the functions are nat, that means that also in the case the difference is negative they will return a 0!

lemma *bhv2cnf-ge-lActive*[*simp*]:
shows ${}_c\uparrow_t(n) \geq \langle c \wedge t\rangle$
using *bhv2cnf-def* **by** *simp*

lemma *bhv2cnf-greater-lActive*[*simp*]:
assumes $n > the\text{-}enat(llength\ (\pi_c(inf\text{-}llist\ t))) - 1$
shows ${}_c\uparrow_t(n) > \langle c \wedge t\rangle$
using *assms bhv2cnf-def* **by** *simp*

lemma *bhv2cnf-lActive*[*simp*]:
assumes $\exists\, i.\ \|c\|_{t\ i}$
and *lfinite* $(\pi_c(inf\text{-}llist\ t))$
shows ${}_c\uparrow_t(the\text{-}enat(llength\ (\pi_c(inf\text{-}llist\ t)))) = Suc\ (\langle c \wedge t\rangle)$
proof −
from *assms* **have** $\pi_c(inf\text{-}llist\ t) \neq []_l$ **by** *simp*
hence $llength\ (\pi_c(inf\text{-}llist\ t)) > 0$ **by** (*simp add*: *lnull-def*)
moreover from ‹*lfinite* $(\pi_c(inf\text{-}llist\ t))$› **have** $llength\ (\pi_c(inf\text{-}llist\ t)) \neq \infty$
using *llength-eq-infty-conv-lfinite* **by** *auto*
ultimately have $the\text{-}enat(llength\ (\pi_c(inf\text{-}llist\ t))) > 0$ **using** *enat-0-iff*(*1*) **by** *fastforce*
hence $the\text{-}enat(llength\ (\pi_c(inf\text{-}llist\ t))) - (the\text{-}enat(llength\ (\pi_c(inf\text{-}llist\ t))) - 1) = 1$ **by** *simp*
thus *?thesis* **using** *bhv2cnf-def* **by** *simp*
qed

1.13.3 Relating the Mappings

In the following we provide some properties about the relationship between the two mapping operators.

lemma *bhv2cnf-cnf2bhv*:
assumes $n \geq \langle c \wedge t \rangle$
shows $_c\uparrow_t(_c\downarrow_t(n)) = n$ (**is** *?lhs = ?rhs*)
proof −
have *?lhs* $= \langle c \wedge t \rangle + ((_c\downarrow_t(n)) - (\textit{the-enat}(\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1))$
using *bhv2cnf-def* **by** *simp*
also have $\ldots = \langle c \wedge t \rangle + (((\textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t)))) - 1 + (n - \langle c \wedge t \rangle)) -$
$(\textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1))$ **using** *cnf2bhv-def* **by** *simp*
also have $(\textit{the-enat}(\textit{llength}\ (\pi_c(\textit{inf-llist}\ t)))) - 1 + (n - (\langle c \wedge t \rangle)) -$
$(\textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1) = (\textit{the-enat}(\textit{llength}\ (\pi_c(\textit{inf-llist}\ t)))) - 1 -$
$((\textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t)))) - 1) + (n - (\langle c \wedge t \rangle))$ **by** *simp*
also have $\ldots = n - (\langle c \wedge t \rangle)$ **by** *simp*
also have $(\langle c \wedge t \rangle) + (n - (\langle c \wedge t \rangle)) = (\langle c \wedge t \rangle) + n - \langle c \wedge t \rangle$ **using** *assms* **by** *simp*
also have $\ldots =$ *?rhs* **by** *simp*
finally show *?thesis* **.**
qed

lemma *cnf2bhv-bhv2cnf*:
assumes $n \geq \textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1$
shows $_c\downarrow_t(_c\uparrow_t(n)) = n$ (**is** *?lhs = ?rhs*)
proof −
have *?lhs* $= \textit{the-enat}(\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1 + ((_c\uparrow_t(n)) - (\langle c \wedge t \rangle))$
using *cnf2bhv-def* **by** *simp*
also have $\ldots = \textit{the-enat}(\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1 + (\langle c \wedge t \rangle +$
$(n - (\textit{the-enat}(\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1)) - (\langle c \wedge t \rangle))$ **using** *bhv2cnf-def* **by** *simp*
also have $\langle c \wedge t \rangle + (n - (\textit{the-enat}(\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1)) - (\langle c \wedge t \rangle) =$
$\langle c \wedge t \rangle - (\langle c \wedge t \rangle) + (n - (\textit{the-enat}(\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1))$ **by** *simp*
also have $\ldots = n - (\textit{the-enat}(\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1)$ **by** *simp*
also have $\textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1 + (n - (\textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1)) =$
$n - (\textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1) + (\textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1)$ **by** *simp*
also have $\ldots = n + ((\textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1) -$
$(\textit{the-enat}\ (\textit{llength}\ (\pi_c(\textit{inf-llist}\ t))) - 1))$ **using** *assms* **by** *simp*
also have $\ldots =$ *?rhs* **by** *simp*
finally show *?thesis* **.**
qed

lemma *p2c-mono-c2p*:
assumes $n \geq \langle c \wedge t \rangle$
and $n' \geq {}_c\downarrow_t(n)$
shows $_c\uparrow_t(n') \geq n$
proof −
from $\langle n' \geq {}_c\downarrow_t(n) \rangle$ **have** $_c\uparrow_t(n') \geq {}_c\uparrow_t(_c\downarrow_t(n))$ **using** *bhv2cnf-mono* **by** *simp*
thus *?thesis* **using** *bhv2cnf-cnf2bhv* $\langle n \geq \langle c \wedge t \rangle \rangle$ **by** *simp*
qed

lemma *p2c-mono-c2p-strict*:
assumes $n \geq \langle c \wedge t \rangle$
and $n <_c\uparrow_t(n')$
shows $_c\downarrow_t(n) < n'$
proof (*rule ccontr*)
assume $\neg\ (_c\downarrow_t(n) < n')$

hence $_c\downarrow_t(n) \geq n'$ **by** *simp*
with ‹$n \geq \langle c \wedge t \rangle$› **have** $_c\uparrow_t(nat\ (_c\downarrow_t(n))) \geq {}_c\uparrow_t(n')$
using *bhv2cnf-mono* **by** *simp*
hence $\neg({}_c\uparrow_t(nat\ (_c\downarrow_t(n))) < {}_c\uparrow_t(n'))$ **by** *simp*
with ‹$n \geq \langle c \wedge t \rangle$› **have** $\neg(n < {}_c\uparrow_t(n'))$
using *bhv2cnf-cnf2bhv* **by** *simp*
with *assms* **show** *False* **by** *simp*
qed

lemma *c2p-mono-p2c*:
assumes $n \geq$ *the-enat* (*llength* (π_c(*inf-llist t*))) − *1*
and $n' \geq {}_c\uparrow_t(n)$
shows $_c\downarrow_t(n') \geq n$
proof −
from ‹$n' \geq {}_c\uparrow_t(n)$› **have** $_c\downarrow_t(n') \geq {}_c\downarrow_t({}_c\uparrow_t(n))$ **using** *cnf2bhv-mono* **by** *simp*
thus *?thesis* **using** *cnf2bhv-bhv2cnf* ‹$n \geq$ *the-enat* (*llength* (π_c(*inf-llist t*))) − *1*› **by** *simp*
qed

lemma *c2p-mono-p2c-strict*:
assumes $n \geq$ *the-enat* (*llength* (π_c(*inf-llist t*))) − *1*
and $n <{}_c\downarrow_t(n')$
shows $_c\uparrow_t(n) < n'$
proof (*rule ccontr*)
assume ¬ $({}_c\uparrow_t(n) < n')$
hence $_c\uparrow_t(n) \geq n'$ **by** *simp*
with ‹$n \geq$ *the-enat* (*llength* (π_c(*inf-llist t*))) − *1*› **have** $_c\downarrow_t(nat\ (_c\uparrow_t(n))) \geq {}_c\downarrow_t(n')$
using *cnf2bhv-mono* **by** *simp*
hence $\neg({}_c\downarrow_t(nat\ (_c\uparrow_t(n))) < {}_c\downarrow_t(n'))$ **by** *simp*
with ‹$n \geq$ *the-enat* (*llength* (π_c(*inf-llist t*))) − *1*› **have** $\neg(n < {}_c\downarrow_t(n'))$
using *cnf2bhv-bhv2cnf* **by** *simp*
with *assms* **show** *False* **by** *simp*
qed

end

end

2 A Calculus for Dynamic Architectures

The following theory formalizes our calculus for dynamic architectures [2, 3] and verifies its soundness. The calculus allows to reason about temporal-logic specifications of component behavior in a dynamic setting. The theory is based on our theory of configuration traces and introduces the notion of behavior trace assertion to specify component behavior in a dynamic setting.

theory *Dynamic-Architecture-Calculus*
imports *Configuration-Traces*
begin

2.1 Extended Natural Numbers

We first provide one additional property for extended natural numbers.

lemma *the-enat-mono*[*simp*]:
assumes $m \neq \infty$

and $n \leq m$
shows *the-enat* $n \leq$ *the-enat* m
using *assms(1) assms(2) enat-ile* **by** *fastforce*

2.2 Lazy Lists

Finally, we provide an additional property for lazy lists.

lemma *llength-geq-enat-lfiniteD*: *llength xs* $\leq$ *enat n* $\Longrightarrow$ *lfinite xs*
using *not-lfinite-llength* **by** *force*

context *dynamic-component*
begin

2.3 Dynamic Evaluation of Temporal Operators

In the following we introduce a function to evaluate a behavior trace assertion over a given configuration trace.

type-synonym *'c bta* = (*nat* $\Rightarrow$ *'c*) $\Rightarrow$ *nat* $\Rightarrow$ *bool*

definition *eval*:: *'id* $\Rightarrow$ (*nat* $\Rightarrow$ *cnf*) $\Rightarrow$ (*nat* $\Rightarrow$ *'cmp*) $\Rightarrow$ *nat*
$\Rightarrow$ *'cmp bta* $\Rightarrow$ *bool*
where *eval cid t t' n* $\gamma \equiv$
$(\exists\, i{\geq}n.\ \|cid\|_{t\ i}) \wedge \gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t'))\ (the\text{-}enat(\langle cid\ \#_n\ inf\text{-}llist\ t\rangle)) \vee$
$(\exists\, i.\ \|cid\|_{t\ i}) \wedge (\nexists\, i'.\ i'{\geq}n \wedge \|cid\|_{t\ i'}) \wedge \gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t'))\ ({}_{cid}{\downarrow_t}(n)) \vee$
$(\nexists\, i.\ \|cid\|_{t\ i}) \wedge \gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t'))\ n$

eval takes a component identifier *cid*, a configuration trace t, a behavior trace t', and point in time n and evaluates behavior trace assertion γ as follows:

- If component *cid* is again activated in the future, γ is evaluated at the next point in time where *cid* is active in t.
- If component *cid* is not again activated in the future but it is activated at least once in t, then γ is evaluated at the point in time given by ${}_{cid}{\downarrow_t}n$.
- If component *cid* is never active in t, then γ is evaluated at time point n.

The following proposition evaluates definition *eval* by showing that a behavior trace assertion γ holds over configuration trace t and continuation t' whenever it holds for the concatenation of the corresponding projection with t'.

proposition *eval-corr*:
eval cid t t' 0 $\gamma \longleftrightarrow \gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ 0$
proof
assume *eval cid t t' 0* γ
with *eval-def* **have** $(\exists\, i{\geq}0.\ \|cid\|_{t\ i}) \wedge$
$\gamma\ (lnth\ (\pi_{cid}inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ (the\text{-}enat\ \langle cid\ \#_{enat\ 0}inf\text{-}llist\ t\rangle) \vee$
$(\exists\, i.\ \|cid\|_{t\ i}) \wedge \neg\ (\exists\, i'{\geq}0.\ \|cid\|_{t\ i'}) \wedge \gamma\ (lnth\ (\pi_{cid}inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ ({}_{cid}{\downarrow_t}0) \vee$
$(\nexists\, i.\ \|cid\|_{t\ i}) \wedge \gamma\ (lnth\ (\pi_{cid}inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ 0$ **by** *simp*
thus $\gamma\ (lnth\ (\pi_{cid}inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ 0$
proof
assume $(\exists\, i{\geq}0.\ \|cid\|_{t\ i}) \wedge \gamma\ (lnth\ (\pi_{cid}inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ (the\text{-}enat\ \langle cid\ \#_{enat\ 0}inf\text{-}llist\ t\rangle)$
moreover have *the-enat* $\langle cid\ \#_{enat\ 0}inf\text{-}llist\ t\rangle = 0$ **using** *zero-enat-def* **by** *auto*
ultimately show *?thesis* **by** *simp*

next
assume $(\exists\, i.\ \|cid\|_{t\ i}) \wedge \neg\ (\exists\, i'{\geq}0.\ \|cid\|_{t\ i'}) \wedge \gamma\ (lnth\ (\pi_{cid} inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ ({}_{cid}{\downarrow_t}0) \vee$
$(\nexists\, i.\ \|cid\|_{t\ i}) \wedge \gamma\ (lnth\ (\pi_{cid} inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ 0$
thus *?thesis* **by** *auto*
qed
next
assume $\gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ 0$
show *eval cid t t′ 0* γ
proof *cases*
assume $\exists\, i.\ \|cid\|_{t\ i}$
hence $\exists\, i{\geq}0.\ \|cid\|_{t\ i}$ **by** *simp*
moreover from $\langle\gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ 0\rangle$ **have**
$\gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ (the\text{-}enat(\langle cid\ \#_{enat\ 0}\ inf\text{-}llist\ t\rangle))$
using *zero-enat-def* **by** *auto*
ultimately show *?thesis* **using** *eval-def* **by** *simp*
next
assume $\nexists\, i.\ \|cid\|_{t\ i}$
with $\langle\gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ 0\rangle$ **show** *?thesis* **using** *eval-def* **by** *simp*
qed
qed

2.3.1 Simplification Rules

lemma *validCI-act*[*simp*]:
assumes $\exists\, i{\geq}n.\ \|cid\|_{t\ i}$
and $\gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ (the\text{-}enat(\langle cid\ \#_n\ inf\text{-}llist\ t\rangle))$
shows *eval cid t t′ n* γ
using *assms eval-def* **by** *simp*

lemma *validCI-cont*[*simp*]:
assumes $\exists\, i.\ \|cid\|_{t\ i}$
and $\nexists\, i'.\ i'{\geq}n \wedge \|cid\|_{t\ i'}$
and $\gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ ({}_{cid}{\downarrow_t}(n))$
shows *eval cid t t′ n* γ
using *assms eval-def* **by** *simp*

lemma *validCI-not-act*[*simp*]:
assumes $\nexists\, i.\ \|cid\|_{t\ i}$
and $\gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ n$
shows *eval cid t t′ n* γ
using *assms eval-def* **by** *simp*

lemma *validCE-act*[*simp*]:
assumes $\exists\, i{\geq}n.\ \|cid\|_{t\ i}$
and *eval cid t t′ n* γ
shows $\gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ (the\text{-}enat(\langle cid\ \#_n\ inf\text{-}llist\ t\rangle))$
using *assms eval-def* **by** *auto*

lemma *validCE-cont*[*simp*]:
assumes $\exists\, i.\ \|cid\|_{t\ i}$
and $\nexists\, i'.\ i'{\geq}n \wedge \|cid\|_{t\ i'}$
and *eval cid t t′ n* γ
shows $\gamma\ (lnth\ ((\pi_{cid}(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ ({}_{cid}{\downarrow_t}(n))$
using *assms eval-def* **by** *auto*

lemma *validCE-not-act*[*simp*]:

assumes $\nexists\, i.\ \|cid\|_{t\ i}$
and *eval cid t t′ n γ*
shows γ (*lnth* ((π_{cid}(*inf-llist t*)) @$_l$ (*inf-llist t′*))) *n*
using *assms eval-def* **by** *auto*

2.3.2 No Activations

proposition *validity1*:
assumes $n \leq n'$
and $\exists\, i \geq n'.\ \|c\|_{t\ i}$
and $\forall\, k \geq n.\ k < n' \longrightarrow \neg\ \|c\|_{t\ k}$
shows *eval c t t′ n γ* $\Longrightarrow$ *eval c t t′ n′ γ*
proof −
assume *eval c t t′ n γ*
moreover from *assms* **have** $\exists\, i \geq n.\ \|c\|_{t\ i}$ **by** (*meson order.trans*)
ultimately have γ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t′*))) (*the-enat* (⟨*c* #$_{enat\ n}$ *inf-llist t*⟩))
using *validCE-act* **by** *blast*
moreover have *enat n′ − 1 < llength* (*inf-llist t*) **by** (*simp add*: *one-enat-def*)
with *assms* **have** *the-enat* (⟨*c* #$_{enat\ n}$ *inf-llist t*⟩) = *the-enat* (⟨*c* #$_{enat\ n'}$ *inf-llist t*⟩)
using *nAct-not-active-same*[*of n n′ inf-llist t c*] **by** *simp*
ultimately have γ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t′*))) (*the-enat* (⟨*c* #$_{enat\ n'}$ *inf-llist t*⟩))
by *simp*
with *assms* **show** *?thesis* **using** *validCI-act* **by** *blast*
qed

proposition *validity2*:
assumes $n \leq n'$
and $\exists\, i \geq n'.\ \|c\|_{t\ i}$
and $\forall\, k \geq n.\ k < n' \longrightarrow \neg\ \|c\|_{t\ k}$
shows *eval c t t′ n′ γ* $\Longrightarrow$ *eval c t t′ n γ*
proof −
assume *eval c t t′ n′ γ*
with ‹$\exists\, i \geq n'.\ \|c\|_{t\ i}$›
have γ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t′*))) (*the-enat* (⟨*c* #$_{enat\ n'}$ *inf-llist t*⟩))
using *validCE-act* **by** *blast*
moreover have *enat n′ − 1 < llength* (*inf-llist t*) **by** (*simp add*: *one-enat-def*)
with *assms* **have** *the-enat* (⟨*c* #$_{enat\ n}$ *inf-llist t*⟩) = *the-enat* (⟨*c* #$_{enat\ n'}$ *inf-llist t*⟩)
using *nAct-not-active-same* **by** *simp*
ultimately have γ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t′*))) (*the-enat* (⟨*c* #$_{enat\ n}$ *inf-llist t*⟩))
by *simp*
moreover from *assms* **have** $\exists\, i \geq n.\ \|c\|_{t\ i}$ **by** (*meson order.trans*)
ultimately show *?thesis* **using** *validCI-act* **by** *blast*
qed

2.4 Specification Operators

In the following we introduce some basic operators for behavior trace assertions.

2.4.1 Predicates

Every predicate can be transformed to a behavior trace assertion.

definition *pred* :: *bool* ⇒ (′*cmp bta*)
where *pred P* ≡ λ *t n*. *P*

lemma *predI*[*intro*]:

fixes *cid t t′ n P*
assumes *P*
shows *eval cid t t′ n* (*pred P*)
proof *cases*
assume (∃ *i*. ‖*cid*‖$_{t\ i}$)
show *?thesis*
proof *cases*
assume ∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$
with *assms* **show** *?thesis* **using** *eval-def pred-def* **by** *auto*
next
assume ¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)
with *assms* **show** *?thesis* **using** *eval-def pred-def* **by** *auto*
qed
next
assume ¬(∃ *i*. ‖*cid*‖$_{t\ i}$)
with *assms* **show** *?thesis* **using** *eval-def pred-def* **by** *auto*
qed

lemma *predE*[*elim*]:
fixes *cid t t′ n P*
assumes *eval cid t t′ n* (*pred P*)
shows *P*
proof *cases*
assume (∃ *i*. ‖*cid*‖$_{t\ i}$)
show *?thesis*
proof *cases*
assume ∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$
with *assms* **show** *?thesis* **using** *eval-def pred-def* **by** *auto*
next
assume ¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)
with *assms* **show** *?thesis* **using** *eval-def pred-def* **by** *auto*
qed
next
assume ¬(∃ *i*. ‖*cid*‖$_{t\ i}$)
with *assms* **show** *?thesis* **using** *eval-def pred-def* **by** *auto*
qed

2.4.2 True and False

abbreviation *true* :: *′cmp bta*
where *true* ≡ λ*t n*. *HOL.True*

abbreviation *false* :: *′cmp bta*
where *false* ≡ λ*t n*. *HOL.False*

2.4.3 Implication

definition *imp* :: (*′cmp bta*) ⇒ (*′cmp bta*) ⇒ (*′cmp bta*) (**infixl** $\longrightarrow^b$ *10*)
where *γ* $\longrightarrow^b$ *γ′* ≡ λ *t n*. *γ t n* ⟶ *γ′ t n*

lemma *impI*[*intro*!]:
assumes *eval cid t t′ n γ* ⟶ *eval cid t t′ n γ′*
shows *eval cid t t′ n* (*γ* $\longrightarrow^b$ *γ′*)
proof *cases*
assume ∃ *i*. ‖*cid*‖$_{t\ i}$
show *?thesis*

proof *cases*
assume $\exists\, i{\geq}n.\ \|cid\|_{t\ i}$
with ‹*eval cid t t′ n γ* ⟶ *eval cid t t′ n γ′*›
have γ (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
⟶ γ' (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
using *eval-def* **by** *blast*
with ‹$\exists\, i{\geq}n.\ \|cid\|_{t\ i}$› **have** *eval cid t t′ n* ($\lambda t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$)
using *validCI-act*[**where** $\gamma{=}\lambda\ t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$] **by** *blast*
thus *?thesis* **using** *imp-def* **by** *simp*
next
assume ¬ ($\exists\, i{\geq}n.\ \|cid\|_{t\ i}$)
with ‹$\exists\, i.\ \|cid\|_{t\ i}$› ‹*eval cid t t′ n γ* ⟶ *eval cid t t′ n γ′*›
have γ (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) ($_{cid}{\downarrow_t}n$)
⟶ γ' (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) ($_{cid}{\downarrow_t}n$) **using** *eval-def* **by** *blast*
with ‹$\exists\, i.\ \|cid\|_{t\ i}$› ‹¬ ($\exists\, i{\geq}n.\ \|cid\|_{t\ i}$)› **have** *eval cid t t′ n* ($\lambda t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$)
using *validCI-cont*[**where** $\gamma{=}\lambda\ t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$] **by** *blast*
thus *?thesis* **using** *imp-def* **by** *simp*
qed
next
assume ¬($\exists\, i.\ \|cid\|_{t\ i}$)
with ‹*eval cid t t′ n γ* ⟶ *eval cid t t′ n γ′*›
have γ (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) *n* ⟶ γ' (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) *n*
using *eval-def* **by** *blast*
with ‹¬($\exists\, i.\ \|cid\|_{t\ i}$)› **have** *eval cid t t′ n* ($\lambda t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$)
using *validCI-not-act*[**where** $\gamma{=}\lambda\ t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$] **by** *blast*
thus *?thesis* **using** *imp-def* **by** *simp*
qed

lemma *impE*[*elim*!]:
assumes *eval cid t t′ n* ($\gamma \longrightarrow^b \gamma'$)
shows *eval cid t t′ n γ* ⟶ *eval cid t t′ n γ′*
proof *cases*
assume ($\exists\, i.\ \|cid\|_{t\ i}$)
show *?thesis*
proof *cases*
assume $\exists\, i{\geq}n.\ \|cid\|_{t\ i}$
moreover from ‹*eval cid t t′ n* ($\gamma \longrightarrow^b \gamma'$)› **have** *eval cid t t′ n* ($\lambda t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$)
using *imp-def* **by** *simp*
ultimately have γ (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
⟶ γ' (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
using *validCE-act*[**where** $\gamma{=}\lambda\ t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$] **by** *blast*
with ‹$\exists\, i{\geq}n.\ \|cid\|_{t\ i}$› **show** *?thesis* **using** *eval-def* **by** *blast*
next
assume ¬ ($\exists\, i{\geq}n.\ \|cid\|_{t\ i}$)
moreover from ‹*eval cid t t′ n* ($\gamma \longrightarrow^b \gamma'$)› **have** *eval cid t t′ n* ($\lambda t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$)
using *imp-def* **by** *simp*
ultimately have γ (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) ($_{cid}{\downarrow_t}n$)
⟶ γ' (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) ($_{cid}{\downarrow_t}n$)
using *validCE-cont*[**where** $\gamma{=}\lambda\ t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$] ‹$\exists\, i.\ \|cid\|_{t\ i}$› **by** *blast*
with ‹¬ ($\exists\, i{\geq}n.\ \|cid\|_{t\ i}$)› ‹$\exists\, i.\ \|cid\|_{t\ i}$› **show** *?thesis* **using** *eval-def* **by** *blast*
qed
next
assume ¬($\exists\, i.\ \|cid\|_{t\ i}$)
moreover from ‹*eval cid t t′ n* ($\gamma \longrightarrow^b \gamma'$)› **have** *eval cid t t′ n* ($\lambda t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$)
using *imp-def* **by** *simp*

ultimately have $\gamma\ (lnth\ (\pi_{cid} inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ n$
$\longrightarrow \gamma'\ (lnth\ (\pi_{cid} inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ n$
using *validCE-not-act*[**where** $\gamma{=}\lambda\ t\ n.\ \gamma\ t\ n \longrightarrow \gamma'\ t\ n$] **by** *blast*
with ‹$\neg(\exists\, i.\ \|cid\|_{t\ i})$› **show** *?thesis* **using** *eval-def* **by** *blast*
qed

2.4.4 Disjunction

definition *disj* :: $('cmp\ bta) \Rightarrow ('cmp\ bta) \Rightarrow ('cmp\ bta)$ (**infixl** $\vee^b$ *15*)
where $\gamma \vee^b \gamma' \equiv \lambda\ t\ n.\ \gamma\ t\ n \vee \gamma'\ t\ n$

lemma *disjI*[*intro*!]:
assumes *eval cid t t' n* $\gamma \vee$ *eval cid t t' n* γ'
shows *eval cid t t' n* $(\gamma \vee^b \gamma')$
using *assms disj-def eval-def* **by** *auto*

lemma *disjE*[*elim*!]:
assumes *eval cid t t' n* $(\gamma \vee^b \gamma')$
shows *eval cid t t' n* $\gamma \vee$ *eval cid t t' n* γ'
proof *cases*
assume $(\exists\, i.\ \|cid\|_{t\ i})$
show *?thesis*
proof *cases*
assume $\exists\, i{\geq}n.\ \|cid\|_{t\ i}$
moreover from ‹*eval cid t t' n* $(\gamma \vee^b \gamma')$› **have** *eval cid t t' n* $(\lambda t\ n.\ \gamma\ t\ n \vee \gamma'\ t\ n)$
using *disj-def* **by** *simp*
ultimately have $\gamma\ (lnth\ (\pi_{cid} inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ (the\text{-}enat\ \langle cid\ \#_{enat\ n} inf\text{-}llist\ t\rangle)$
$\vee\ \gamma'\ (lnth\ (\pi_{cid} inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ (the\text{-}enat\ \langle cid\ \#_{enat\ n} inf\text{-}llist\ t\rangle)$
using *validCE-act*[**where** $\gamma{=}\lambda\ t\ n.\ \gamma\ t\ n \vee \gamma'\ t\ n$] **by** *blast*
with ‹$\exists\, i{\geq}n.\ \|cid\|_{t\ i}$› **show** *?thesis*
using *validCI-act*[*of n cid t* $\gamma\ t'$] *validCI-act*[*of n cid t* $\gamma'\ t'$] **by** *blast*
next
assume $\neg\ (\exists\, i{\geq}n.\ \|cid\|_{t\ i})$
moreover from ‹*eval cid t t' n* $(\gamma \vee^b \gamma')$› **have** *eval cid t t' n* $(\lambda t\ n.\ \gamma\ t\ n \vee \gamma'\ t\ n)$
using *disj-def* **by** *simp*
ultimately have $\gamma\ (lnth\ (\pi_{cid} inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ ({}_{cid}{\downarrow_t}n)$
$\vee\ \gamma'\ (lnth\ (\pi_{cid} inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ ({}_{cid}{\downarrow_t}n)$
using *validCE-cont*[**where** $\gamma{=}\lambda\ t\ n.\ \gamma\ t\ n \vee \gamma'\ t\ n$] ‹$\exists\, i.\ \|cid\|_{t\ i}$› **by** *blast*
with ‹$\neg\ (\exists\, i{\geq}n.\ \|cid\|_{t\ i})$› ‹$\exists\, i.\ \|cid\|_{t\ i}$› **show** *?thesis*
using *validCI-cont*[*of cid t n* $\gamma\ t'$] *validCI-cont*[*of cid t n* $\gamma'\ t'$] **by** *blast*
qed
next
assume $\neg(\exists\, i.\ \|cid\|_{t\ i})$
moreover from ‹*eval cid t t' n* $(\gamma \vee^b \gamma')$› **have** *eval cid t t' n* $(\lambda t\ n.\ \gamma\ t\ n \vee \gamma'\ t\ n)$
using *disj-def* **by** *simp*
ultimately have $\gamma\ (lnth\ (\pi_{cid} inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ n$
$\vee\ \gamma'\ (lnth\ (\pi_{cid} inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ n$
using *validCE-not-act*[**where** $\gamma{=}\lambda\ t\ n.\ \gamma\ t\ n \vee \gamma'\ t\ n$] **by** *blast*
with ‹$\neg(\exists\, i.\ \|cid\|_{t\ i})$› **show** *?thesis*
using *validCI-not-act*[*of cid t* $\gamma\ t'\ n$] *validCI-not-act*[*of cid t* $\gamma'\ t'\ n$] **by** *blast*
qed

2.4.5 Conjunction

definition *conj* :: $('cmp\ bta) \Rightarrow ('cmp\ bta) \Rightarrow ('cmp\ bta)$ (**infixl** $\wedge^b$ *20*)
where $\gamma \wedge^b \gamma' \equiv \lambda\ t\ n.\ \gamma\ t\ n \wedge \gamma'\ t\ n$

lemma *conjI*[*intro*!]:
assumes *eval cid t t′ n γ* ∧ *eval cid t t′ n γ′*
shows *eval cid t t′ n* (*γ* ∧^{b} *γ′*)
proof *cases*
assume ∃ *i*. ‖*cid*‖$_{t\ i}$
show *?thesis*
proof *cases*
assume ∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$
with ‹*eval cid t t′ n γ* ∧ *eval cid t t′ n γ′*›
have *γ* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
∧ *γ′* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
using *eval-def* **by** *blast*
with ‹∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$› **have** *eval cid t t′ n* (λ*t n*. *γ t n* ∧ *γ′ t n*)
using *validCI-act*[**where** *γ*=λ *t n*. *γ t n* ∧ *γ′ t n*] **by** *blast*
thus *?thesis* **using** *conj-def* **by** *simp*
next
assume ¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)
with ‹∃ *i*. ‖*cid*‖$_{t\ i}$› ‹*eval cid t t′ n γ* ∧ *eval cid t t′ n γ′*›
have *γ* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*cid*↓$_t$*n*)
∧ *γ′* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*cid*↓$_t$*n*) **using** *eval-def* **by** *blast*
with ‹∃ *i*. ‖*cid*‖$_{t\ i}$› ‹¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)› **have** *eval cid t t′ n* (λ*t n*. *γ t n* ∧ *γ′ t n*)
using *validCI-cont*[**where** *γ*=λ *t n*. *γ t n* ∧ *γ′ t n*] **by** *blast*
thus *?thesis* **using** *conj-def* **by** *simp*
qed
next
assume ¬(∃ *i*. ‖*cid*‖$_{t\ i}$)
with ‹*eval cid t t′ n γ* ∧ *eval cid t t′ n γ′*› **have** *γ* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) *n*
∧ *γ′* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) *n* **using** *eval-def* **by** *blast*
with ‹¬(∃ *i*. ‖*cid*‖$_{t\ i}$)› **have** *eval cid t t′ n* (λ*t n*. *γ t n* ∧ *γ′ t n*)
using *validCI-not-act*[**where** *γ*=λ *t n*. *γ t n* ∧ *γ′ t n*] **by** *blast*
thus *?thesis* **using** *conj-def* **by** *simp*
qed

lemma *conjE*[*elim*!]:
assumes *eval cid t t′ n* (*γ* ∧^{b} *γ′*)
shows *eval cid t t′ n γ* ∧ *eval cid t t′ n γ′*
proof *cases*
assume (∃ *i*. ‖*cid*‖$_{t\ i}$)
show *?thesis*
proof *cases*
assume ∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$
moreover from ‹*eval cid t t′ n* (*γ* ∧^{b} *γ′*)› **have** *eval cid t t′ n* (λ*t n*. *γ t n* ∧ *γ′ t n*)
using *conj-def* **by** *simp*
ultimately have *γ* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
∧ *γ′* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
using *validCE-act*[**where** *γ*=λ *t n*. *γ t n* ∧ *γ′ t n*] **by** *blast*
with ‹∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$› **show** *?thesis* **using** *eval-def* **by** *blast*
next
assume ¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)
moreover from ‹*eval cid t t′ n* (*γ* ∧^{b} *γ′*)› **have** *eval cid t t′ n* (λ*t n*. *γ t n* ∧ *γ′ t n*)
using *conj-def* **by** *simp*
ultimately have *γ* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*cid*↓$_t$*n*)
∧ *γ′* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*cid*↓$_t$*n*)
using *validCE-cont*[**where** *γ*=λ *t n*. *γ t n* ∧ *γ′ t n*] ‹∃ *i*. ‖*cid*‖$_{t\ i}$› **by** *blast*

with ‹¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)› ‹∃ *i*. ‖*cid*‖$_{t\ i}$› **show** *?thesis* **using** *eval-def* **by** *blast*
qed
next
assume ¬(∃ *i*. ‖*cid*‖$_{t\ i}$)
moreover from ‹*eval cid t t′ n* (*γ* ∧^{b} *γ′*)› **have** *eval cid t t′ n* (λ*t n*. *γ t n* ∧ *γ′ t n*)
using *conj-def* **by** *simp*
ultimately have *γ* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) *n* ∧ *γ′* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) *n*
using *validCE-not-act*[**where** *γ*=λ *t n*. *γ t n* ∧ *γ′ t n*] **by** *blast*
with ‹¬(∃ *i*. ‖*cid*‖$_{t\ i}$)› **show** *?thesis* **using** *eval-def* **by** *blast*
qed

2.4.6 Negation

definition *neg* :: (*'cmp bta*) ⇒ (*'cmp bta*) (¬^{b} - [*19*] *19*)
where ¬^{b} *γ* ≡ λ *t n*. ¬ *γ t n*

lemma *negI*[*intro*!]:
assumes ¬ *eval cid t t′ n γ*
shows *eval cid t t′ n* (¬^{b} *γ*)
proof *cases*
assume ∃ *i*. ‖*cid*‖$_{t\ i}$
show *?thesis*
proof *cases*
assume ∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$
with ‹¬ *eval cid t t′ n γ*›
have ¬ *γ* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
using *eval-def* **by** *blast*
with ‹∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$› **have** *eval cid t t′ n* (λ*t n*. ¬ *γ t n*)
using *validCI-act*[**where** *γ*=λ *t n*. ¬ *γ t n*] **by** *blast*
thus *?thesis* **using** *neg-def* **by** *simp*
next
assume ¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)
with ‹∃ *i*. ‖*cid*‖$_{t\ i}$› ‹¬ *eval cid t t′ n γ*›
have ¬ *γ* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*cid*↓$_t$*n*) **using** *eval-def* **by** *blast*
with ‹∃ *i*. ‖*cid*‖$_{t\ i}$› ‹¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)› **have** *eval cid t t′ n* (λ*t n*. ¬ *γ t n*)
using *validCI-cont*[**where** *γ*=λ *t n*. ¬ *γ t n*] **by** *blast*
thus *?thesis* **using** *neg-def* **by** *simp*
qed
next
assume ¬(∃ *i*. ‖*cid*‖$_{t\ i}$)
with ‹¬ *eval cid t t′ n γ*› **have** ¬ *γ* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) *n* **using** *eval-def* **by** *blast*
with ‹¬(∃ *i*. ‖*cid*‖$_{t\ i}$)› **have** *eval cid t t′ n* (λ*t n*. ¬ *γ t n*)
using *validCI-not-act*[**where** *γ*=λ *t n*. ¬ *γ t n*] **by** *blast*
thus *?thesis* **using** *neg-def* **by** *simp*
qed

lemma *negE*[*elim*!]:
assumes *eval cid t t′ n* (¬^{b} *γ*)
shows ¬ *eval cid t t′ n γ*
proof *cases*
assume (∃ *i*. ‖*cid*‖$_{t\ i}$)
show *?thesis*
proof *cases*
assume ∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$
moreover from ‹*eval cid t t′ n* (¬^{b} *γ*)› **have** *eval cid t t′ n* (λ*t n*. ¬ *γ t n*) **using** *neg-def* **by** *simp*
ultimately have ¬ *γ* (*lnth* (*π*$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)

using *validCE-act*[**where** γ=λ *t n*. $\neg$ γ *t n*] **by** *blast*
with ‹$\exists$ *i*$\geq$*n*. $\|cid\|_{t\ i}$› **show** *?thesis* **using** *eval-def* **by** *blast*
next
assume $\neg$ ($\exists$ *i*$\geq$*n*. $\|cid\|_{t\ i}$)
moreover from ‹*eval cid t t′ n* ($\neg^b$ γ)› **have** *eval cid t t′ n* (λ*t n*. $\neg$ γ *t n*) **using** *neg-def* **by** *simp*
ultimately have $\neg$ γ (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) ($_{cid}\downarrow_t$*n*)
using *validCE-cont*[**where** γ=λ *t n*. $\neg$ γ *t n*] ‹$\exists$ *i*. $\|cid\|_{t\ i}$› **by** *blast*
with ‹$\neg$ ($\exists$ *i*$\geq$*n*. $\|cid\|_{t\ i}$)› ‹$\exists$ *i*. $\|cid\|_{t\ i}$› **show** *?thesis* **using** *eval-def* **by** *blast*
qed
next
assume $\neg$($\exists$ *i*. $\|cid\|_{t\ i}$)
moreover from ‹*eval cid t t′ n* ($\neg^b$ γ)› **have** *eval cid t t′ n* (λ*t n*. $\neg$ γ *t n*) **using** *neg-def* **by** *simp*
ultimately have $\neg$ γ (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) *n*
using *validCE-not-act*[**where** γ=λ *t n*. $\neg$ γ *t n*] **by** *blast*
with ‹$\neg$($\exists$ *i*. $\|cid\|_{t\ i}$)› **show** *?thesis* **using** *eval-def* **by** *blast*
qed

2.4.7 Quantifiers

definition *all* :: (*′a* $\Rightarrow$ (*′cmp bta*))
$\Rightarrow$ (*′cmp bta*) (**binder** $\forall_b$ *10*)
where *all P* $\equiv$ λ*t n*. ($\forall$ *y*. (*P y t n*))

lemma *allI*[*intro*!]:
assumes $\forall$ *p*. *eval cid t t′ n* (γ *p*)
shows *eval cid t t′ n* (*all* (λ*p*. γ *p*))
proof *cases*
assume $\exists$ *i*. $\|cid\|_{t\ i}$
show *?thesis*
proof *cases*
assume $\exists$ *i*$\geq$*n*. $\|cid\|_{t\ i}$
with ‹$\forall$ *p*. *eval cid t t′ n* (γ *p*)›
have $\forall$ *p*. (γ *p*) (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat}$ *n*|*inf-llist t*⟩)
using *eval-def* **by** *blast*
with ‹$\exists$ *i*$\geq$*n*. $\|cid\|_{t\ i}$› **have** *eval cid t t′ n* (λ*t n*. ($\forall$ *y*. (γ *y t n*)))
using *validCI-act*[**where** γ=λ*t n*. ($\forall$ *y*. (γ *y t n*))] **by** *blast*
thus *?thesis* **using** *all-def*[*of* γ] **by** *auto*
next
assume $\neg$ ($\exists$ *i*$\geq$*n*. $\|cid\|_{t\ i}$)
with ‹$\exists$ *i*. $\|cid\|_{t\ i}$› ‹$\forall$ *p*. *eval cid t t′ n* (γ *p*)›
have $\forall$ *p*. (γ *p*) (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) ($_{cid}\downarrow_t$*n*)
using *eval-def* **by** *blast*
with ‹$\exists$ *i*. $\|cid\|_{t\ i}$› ‹$\neg$ ($\exists$ *i*$\geq$*n*. $\|cid\|_{t\ i}$)› **have** *eval cid t t′ n* (λ*t n*. ($\forall$ *y*. (γ *y t n*)))
using *validCI-cont*[**where** γ=λ*t n*. ($\forall$ *y*. (γ *y t n*))] **by** *blast*
thus *?thesis* **using** *all-def*[*of* γ] **by** *auto*
qed
next
assume $\neg$($\exists$ *i*. $\|cid\|_{t\ i}$)
with ‹$\forall$ *p*. *eval cid t t′ n* (γ *p*)› **have** $\forall$ *p*. (γ *p*) (*lnth* (π_{cid}*inf-llist t* @$_l$ *inf-llist t′*)) *n*
using *eval-def* **by** *blast*
with ‹$\neg$($\exists$ *i*. $\|cid\|_{t\ i}$)› **have** *eval cid t t′ n* (λ*t n*. ($\forall$ *y*. (γ *y t n*)))
using *validCI-not-act*[**where** γ=λ*t n*. ($\forall$ *y*. (γ *y t n*))] **by** *blast*
thus *?thesis* **using** *all-def*[*of* γ] **by** *auto*
qed

lemma *allE*[*elim*!]:

assumes *eval cid t t′ n* (*all* (λ*p*. γ *p*))
shows ∀ *p*. *eval cid t t′ n* (γ *p*)
proof *cases*
assume (∃ *i*. ‖*cid*‖$_{t\ i}$)
show *?thesis*
proof *cases*
assume ∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$
moreover from ‹*eval cid t t′ n* (*all* (λ*p*. γ *p*))› **have** *eval cid t t′ n* (λ*t n*. (∀ *y*. (γ *y t n*)))
using *all-def*[*of* γ] **by** *auto*
ultimately have ∀ *p*. (γ *p*) (*lnth* (π$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
using *validCE-act*[**where** γ=λ*t n*. (∀ *y*. (γ *y t n*))] **by** *blast*
with ‹∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$› **show** *?thesis* **using** *eval-def* **by** *blast*
next
assume ¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)
moreover from ‹*eval cid t t′ n* (*all* (λ*p*. γ *p*))› **have** *eval cid t t′ n* (λ*t n*. (∀ *y*. (γ *y t n*)))
using *all-def*[*of* γ] **by** *auto*
ultimately have ∀ *p*. (γ *p*) (*lnth* (π$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*cid*↓$_t$*n*)
using *validCE-cont*[**where** γ=λ*t n*. (∀ *y*. (γ *y t n*))] ‹∃ *i*. ‖*cid*‖$_{t\ i}$› **by** *blast*
with ‹¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)› ‹∃ *i*. ‖*cid*‖$_{t\ i}$› **show** *?thesis* **using** *eval-def* **by** *blast*
qed
next
assume ¬(∃ *i*. ‖*cid*‖$_{t\ i}$)
moreover from ‹*eval cid t t′ n* (*all* (λ*p*. γ *p*))› **have** *eval cid t t′ n* (λ*t n*. (∀ *y*. (γ *y t n*)))
using *all-def*[*of* γ] **by** *auto*
ultimately have ∀ *p*. (γ *p*) (*lnth* (π$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) *n*
using *validCE-not-act*[**where** γ=λ*t n*. (∀ *y*. (γ *y t n*))] **by** *blast*
with ‹¬(∃ *i*. ‖*cid*‖$_{t\ i}$)› **show** *?thesis* **using** *eval-def* **by** *blast*
qed

definition *ex* :: (′*a* ⇒ (′*cmp bta*))
⇒ (′*cmp bta*) (**binder** ∃$_b$ *10*)
where *ex P* ≡ λ*t n*. (∃ *y*. (*P y t n*))

lemma *exI*[*intro*!]:
assumes ∃ *p*. *eval cid t t′ n* (γ *p*)
shows *eval cid t t′ n* (∃$_b$*p*. γ *p*)
proof *cases*
assume ∃ *i*. ‖*cid*‖$_{t\ i}$
show *?thesis*
proof *cases*
assume ∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$
with ‹∃ *p*. *eval cid t t′ n* (γ *p*)›
have ∃ *p*. (γ *p*) (*lnth* (π$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*the-enat* ⟨*cid* #$_{enat\ n}$*inf-llist t*⟩)
using *eval-def* **by** *blast*
with ‹∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$› **have** *eval cid t t′ n* (λ*t n*. (∃ *y*. (γ *y t n*)))
using *validCI-act*[**where** γ=λ*t n*. (∃ *y*. (γ *y t n*))] **by** *blast*
thus *?thesis* **using** *ex-def*[*of* γ] **by** *auto*
next
assume ¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)
with ‹∃ *i*. ‖*cid*‖$_{t\ i}$› ‹∃ *p*. *eval cid t t′ n* (γ *p*)›
have ∃ *p*. (γ *p*) (*lnth* (π$_{cid}$*inf-llist t* @$_l$ *inf-llist t′*)) (*cid*↓$_t$*n*) **using** *eval-def* **by** *blast*
with ‹∃ *i*. ‖*cid*‖$_{t\ i}$› ‹¬ (∃ *i*≥*n*. ‖*cid*‖$_{t\ i}$)› **have** *eval cid t t′ n* (λ*t n*. (∃ *y*. (γ *y t n*)))
using *validCI-cont*[**where** γ=λ*t n*. (∃ *y*. (γ *y t n*))] **by** *blast*
thus *?thesis* **using** *ex-def*[*of* γ] **by** *auto*
qed

next
assume $\neg(\exists\, i.\ \|cid\|_{t\ i})$
with ‹$\exists\, p.$ *eval cid t t′ n* $(\gamma\ p)$› **have** $\exists\, p.\ (\gamma\ p)$ (*lnth* $(\pi_{cid}$*inf-llist t* $@_l$ *inf-llist t′*)) *n*
using *eval-def* **by** *blast*
with ‹$\neg(\exists\, i.\ \|cid\|_{t\ i})$› **have** *eval cid t t′ n* $(\lambda t\ n.\ (\exists\, y.\ (\gamma\ y\ t\ n)))$
using *validCI-not-act*[**where** $\gamma{=}\lambda t\ n.\ (\exists\, y.\ (\gamma\ y\ t\ n))$] **by** *blast*
thus *?thesis* **using** *ex-def*[*of* γ] **by** *auto*
qed

lemma *exE*[*elim*!]:
assumes *eval cid t t′ n* $(\exists_b p.\ \gamma\ p)$
shows $\exists\, p.$ *eval cid t t′ n* $(\gamma\ p)$
proof *cases*
assume $(\exists\, i.\ \|cid\|_{t\ i})$
show *?thesis*
proof *cases*
assume $\exists\, i{\geq}n.\ \|cid\|_{t\ i}$
moreover from ‹*eval cid t t′ n* $(ex\ (\lambda p.\ \gamma\ p))$› **have** *eval cid t t′ n* $(\lambda t\ n.\ (\exists\, y.\ (\gamma\ y\ t\ n)))$
using *ex-def*[*of* γ] **by** *auto*
ultimately have $\exists\, p.\ (\gamma\ p)$ (*lnth* $(\pi_{cid}$*inf-llist t* $@_l$ *inf-llist t′*)) (*the-enat* $\langle cid\ \#_{enat}\ n$*inf-llist t*$\rangle$)
using *validCE-act*[**where** $\gamma{=}\lambda t\ n.\ (\exists\, y.\ (\gamma\ y\ t\ n))$] **by** *blast*
with ‹$\exists\, i{\geq}n.\ \|cid\|_{t\ i}$› **show** *?thesis* **using** *eval-def* **by** *blast*
next
assume $\neg\ (\exists\, i{\geq}n.\ \|cid\|_{t\ i})$
moreover from ‹*eval cid t t′ n* $(\exists_b p.\ \gamma\ p)$› **have** *eval cid t t′ n* $(\lambda t\ n.\ (\exists\, y.\ (\gamma\ y\ t\ n)))$
using *ex-def*[*of* γ] **by** *auto*
ultimately have $\exists\, p.\ (\gamma\ p)$ (*lnth* $(\pi_{cid}$*inf-llist t* $@_l$ *inf-llist t′*)) $({}_{cid}{\downarrow}_t n)$
using *validCE-cont*[**where** $\gamma{=}\lambda t\ n.\ (\exists\, y.\ (\gamma\ y\ t\ n))$] ‹$\exists\, i.\ \|cid\|_{t\ i}$› **by** *blast*
with ‹$\neg\ (\exists\, i{\geq}n.\ \|cid\|_{t\ i})$› ‹$\exists\, i.\ \|cid\|_{t\ i}$› **show** *?thesis* **using** *eval-def* **by** *blast*
qed
next
assume $\neg(\exists\, i.\ \|cid\|_{t\ i})$
moreover from ‹*eval cid t t′ n* $(\exists_b p.\ \gamma\ p)$› **have** *eval cid t t′ n* $(\lambda t\ n.\ (\exists\, y.\ (\gamma\ y\ t\ n)))$
using *ex-def*[*of* γ] **by** *auto*
ultimately have $\exists\, p.\ (\gamma\ p)$ (*lnth* $(\pi_{cid}$*inf-llist t* $@_l$ *inf-llist t′*)) *n*
using *validCE-not-act*[**where** $\gamma{=}\lambda t\ n.\ (\exists\, y.\ (\gamma\ y\ t\ n))$] **by** *blast*
with ‹$\neg(\exists\, i.\ \|cid\|_{t\ i})$› **show** *?thesis* **using** *eval-def* **by** *blast*
qed

2.4.8 Behavior Assertions

First we provide rules for basic behavior assertions.

definition *ba* :: $('cmp \Rightarrow bool) \Rightarrow ('cmp\ bta)\ ([\text{-}]_b)$
where *ba* $\varphi \equiv \lambda\ t\ n.\ \varphi\ (t\ n)$

lemma *baIA*[*intro*]:
fixes *c*::*'id*
and *t*::*nat* $\Rightarrow$ *cnf*
and *t′*::*nat* $\Rightarrow$ *'cmp*
and *n*::*nat*
assumes $\exists\, i{\geq}n.\ \|c\|_{t\ i}$
and $\varphi\ (\sigma_c(t\ \langle c \rightarrow t\rangle_n))$
shows *eval c t t′ n* (*ba* φ)
proof −
from *assms* **have** $\varphi\ (\sigma_c(t\ \langle c \rightarrow t\rangle_n))$ **by** *simp*

moreover have $\sigma_c(t\ \langle c \to t\rangle_n) = lnth\ (\pi_c(inf\text{-}llist\ t))\ (the\text{-}enat\ (\langle c\ \#_{\langle c \to t\rangle_n}\ inf\text{-}llist\ t\rangle))$
proof −
have $enat\ (Suc\ \langle c \to t\rangle_n) < llength\ (inf\text{-}llist\ t)$ **using** *enat-ord-code* **by** *simp*
moreover from *assms* **have** $\|c\|_{t\ (\langle c \to t\rangle_n)}$ **using** *nxtActI* **by** *simp*
hence $\|c\|_{lnth\ (inf\text{-}llist\ t)\ \langle c \to t\rangle_n}$ **by** *simp*
ultimately show *?thesis* **using** *proj-active-nth* **by** *simp*
qed
ultimately have $\varphi\ (lnth\ (\pi_c(inf\text{-}llist\ t))\ (the\text{-}enat(\langle c\ \#_{\langle c \to t\rangle_n}\ inf\text{-}llist\ t\rangle)))$ **by** *simp*
moreover have $\langle c\ \#_n\ inf\text{-}llist\ t\rangle = \langle c\ \#_{\langle c \to t\rangle_n}\ inf\text{-}llist\ t\rangle$
proof −
from *assms* **have** $\nexists k.\ n{\leq}k \wedge k{<}\langle c \to t\rangle_n \wedge \|c\|_{t\ k}$ **using** *nxtActI* **by** *simp*
hence $\neg\ (\exists k{\geq}n.\ k < \langle c \to t\rangle_n \wedge \|c\|_{lnth\ (inf\text{-}llist\ t)\ k})$ **by** *simp*
moreover have $enat\ \langle c \to t\rangle_n - 1 < llength\ (inf\text{-}llist\ t)$ **by** (*simp add*: *one-enat-def*)
moreover from *assms* **have** $\langle c \to t\rangle_n{\geq}n$ **using** *nxtActI* **by** *simp*
ultimately show *?thesis* **using** *nAct-not-active-same*[*of* $n\ \langle c \to t\rangle_n$ *inf-llist t c*] **by** *simp*
qed
ultimately have $\varphi\ (lnth\ (\pi_c(inf\text{-}llist\ t))\ (the\text{-}enat(\langle c\ \#_n\ inf\text{-}llist\ t\rangle)))$ **by** *simp*
moreover have $enat\ (the\text{-}enat\ (\langle c\ \#_{enat\ n}\ inf\text{-}llist\ t\rangle)) < llength\ (\pi_c(inf\text{-}llist\ t))$
proof −
have $ltake\ \infty\ (inf\text{-}llist\ t) = (inf\text{-}llist\ t)$ **using** *ltake-all*[*of inf-llist t*] **by** *simp*
hence $llength\ (\pi_c(inf\text{-}llist\ t)) = \langle c\ \#_\infty\ inf\text{-}llist\ t\rangle$ **using** *nAct-def* **by** *simp*
moreover have $\langle c\ \#_{enat\ n}\ inf\text{-}llist\ t\rangle < \langle c\ \#_\infty\ inf\text{-}llist\ t\rangle$
proof −
have $enat\ \langle c \to t\rangle_n < llength\ (inf\text{-}llist\ t)$ **by** *simp*
moreover from *assms* **have** $\langle c \to t\rangle_n{\geq}n$ **and** $\|c\|_{t\ (\langle c \to t\rangle_n)}$ **using** *nxtActI* **by** *auto*
ultimately show *?thesis* **using** *nAct-less*[*of* $\langle c \to t\rangle_n$ *inf-llist t n* ∞] **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed
hence $lnth\ (\pi_c(inf\text{-}llist\ t))\ (the\text{-}enat\ (\langle c\ \#_n\ inf\text{-}llist\ t\rangle)) =$
$lnth\ ((\pi_c(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t'))\ (the\text{-}enat\ (\langle c\ \#_n\ inf\text{-}llist\ t\rangle))$
using *lnth-lappend1*[*of the-enat* $(\langle c\ \#_{enat\ n}\ inf\text{-}llist\ t\rangle)\ \pi_c(inf\text{-}llist\ t)\ inf\text{-}llist\ t'$] **by** *simp*
ultimately have $\varphi\ (lnth\ ((\pi_c(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t'))\ (the\text{-}enat(\langle c\ \#_n\ inf\text{-}llist\ t\rangle)))$ **by** *simp*
hence $\varphi\ (lnth\ ((\pi_c(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t'))\ (the\text{-}enat\ (\langle c\ \#_n\ inf\text{-}llist\ t\rangle)))$ **by** *simp*
moreover from *assms* **have** $\langle c \to t\rangle_n{\geq}n$ **and** $\|c\|_{t\ (\langle c \to t\rangle_n)}$ **using** *nxtActI* **by** *auto*
ultimately have $(\exists i{\geq}snd\ (t,\ n).\ \|c\|_{fst\ (t,\ n)\ i}) \wedge$
$\varphi\ (lnth\ ((\pi_c(inf\text{-}llist\ (fst\ (t,\ n))))\ @_l\ (inf\text{-}llist\ t'))$
$(the\text{-}enat\ (\langle c\ \#_{the\text{-}enat\ (snd\ (t,n))}\ inf\text{-}llist\ (fst\ (t,\ n))\rangle)))$ **by** *auto*
thus *?thesis* **using** *ba-def* **by** *simp*
qed

lemma *baIN1*[*intro*]:
fixes $c{::}'id$
and $t{::}nat \Rightarrow cnf$
and $t'{::}nat \Rightarrow 'cmp$
and $n{::}nat$
assumes *act*: $\exists i.\ \|c\|_{t\ i}$
and *nAct*: $\nexists i.\ i{\geq}n \wedge \|c\|_{t\ i}$
and *al*: $\varphi\ (t'\ (n - \langle c \wedge t\rangle - 1))$
shows $eval\ c\ t\ t'\ n\ (ba\ \varphi)$
proof −
have $t'\ (n - \langle c \wedge t\rangle - 1) = lnth\ (inf\text{-}llist\ t')\ (n - \langle c \wedge t\rangle - 1)$ **by** *simp*
moreover have $\ldots = lnth\ ((\pi_c(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t'))\ (_c{\downarrow_t}(n))$
using *act nAct cnf2bhv-lnth-lappend* **by** *simp*

ultimately have φ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*)) ($_{c\downarrow_t}$(*n*))) **using** *al* **by** *simp*
with *act nAct* **show** *?thesis* **using** *ba-def* **by** *simp*
qed

lemma *baIN2*[*intro*]:
fixes *c*::*'id*
and *t*::*nat* ⇒ *cnf*
and *t'*::*nat* ⇒ *'cmp*
and *n*::*nat*
assumes *nAct*: ∄ *i*. ‖*c*‖$_{t\ i}$
and *al*: φ (*t' n*)
shows *eval c t t' n* (*ba* φ)
proof −
have *t' n* = *lnth* (*inf-llist t'*) *n* **by** *simp*
moreover have ... = *lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*)) *n*
proof −
from *nAct* **have** π_c(*inf-llist t*) = []$_l$ **by** *simp*
hence (π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*) = *inf-llist t'* **by** (*simp add*: ‹π_c*inf-llist t* = []$_l$›)
thus *?thesis* **by** *simp*
qed
ultimately have φ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*)) *n*) **using** *al* **by** *simp*
hence φ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*)) *n*) **by** *simp*
with *nAct* **show** *?thesis* **using** *ba-def* **by** *simp*
qed

lemma *baIANow*[*intro*]:
fixes *t n c* φ
assumes φ (σ_c(*t n*))
and ‖*c*‖$_{t\ n}$
shows *eval c t t' n* (*ba* φ)
proof −
from *assms* **have** φ(σ_c(*t* ⟨*c* → *t*⟩$_n$)) **using** *nxtAct-active* **by** *simp*
with *assms* **show** *?thesis* **using** *baIA* **by** *blast*
qed

lemma *baEA*[*elim*]:
fixes *c*::*'id*
and *t*::*nat* ⇒ *cnf*
and *t'*::*nat* ⇒ *'cmp*
and *n*::*nat*
and *i*::*nat*
assumes ∃ *i*≥*n*. ‖*c*‖$_{t\ i}$
and *eval c t t' n* (*ba* φ)
shows φ (σ_c(*t* ⟨*c* → *t*⟩$_n$))
proof −
from ‹*eval c t t' n* (*ba* φ)› **have** *eval c t t' n* (λ *t n*. φ (*t n*)) **using** *ba-def* **by** *simp*
moreover from *assms* **have** ⟨*c* → *t*⟩$_n$≥*n* **and** ‖*c*‖$_{t\ (\langle c \to t\rangle_n)}$ **using** *nxtActI*[*of n c t*] **by** *auto*
ultimately have φ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*)) (*the-enat* (⟨*c* #$_n$ *inf-llist t*⟩))))
using *validCE-act* **by** *blast*
hence φ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*)) (*the-enat* (⟨*c* #$_n$ *inf-llist t*⟩))) **by** *simp*
moreover have *enat* (*the-enat* (⟨*c* #$_{enat\ n}$ *inf-llist t*⟩)) < *llength* (π_c(*inf-llist t*))
proof −
have *ltake* ∞ (*inf-llist t*) = (*inf-llist t*) **using** *ltake-all*[*of inf-llist t*] **by** *simp*
hence *llength* (π_c(*inf-llist t*)) = ⟨*c* #$_\infty$ *inf-llist t*⟩ **using** *nAct-def* **by** *simp*
moreover have ⟨*c* #$_{enat\ n}$ *inf-llist t*⟩ < ⟨*c* #$_\infty$ *inf-llist t*⟩

```
    proof −
      have enat ⟨c → t⟩n < llength (inf-llist t) by simp
      with ‹⟨c → t⟩n≥n› ‹‖c‖t ⟨c → t⟩n› show ?thesis using nAct-less by simp
    qed
    ultimately show ?thesis by simp
  qed
  hence lnth ((πc(inf-llist t)) @l (inf-llist t′)) (the-enat (⟨c #n inf-llist t⟩)) =
    lnth (πc(inf-llist t)) (the-enat (⟨c #n inf-llist t⟩))
    using lnth-lappend1[of the-enat (⟨c #enat n inf-llist t⟩) πc(inf-llist t) inf-llist t′] by simp
  ultimately have φ (lnth (πc(inf-llist t)) (the-enat (⟨c #n inf-llist t⟩))) by simp
  moreover have ⟨c #n inf-llist t⟩ = ⟨c #⟨c → t⟩n inf-llist t⟩
  proof −
    from assms have ∄k. n≤k ∧ k<⟨c → t⟩n ∧ ‖c‖t k using nxtActI[of n c t] by auto
    hence ¬ (∃k≥n. k < ⟨c → t⟩n ∧ ‖c‖lnth (inf-llist t) k) by simp
    moreover have enat ⟨c → t⟩n − 1 < llength (inf-llist t) by (simp add: one-enat-def)
    ultimately show ?thesis using ‹⟨c → t⟩n≥n› nAct-not-active-same by simp
  qed
  moreover have σc(t ⟨c → t⟩n) = lnth (πc(inf-llist t)) (the-enat (⟨c #⟨c → t⟩n inf-llist t⟩))
  proof −
    have enat (Suc i) < llength (inf-llist t) using enat-ord-code by simp
    moreover from ‹‖c‖t ⟨c → t⟩n› have ‖c‖lnth (inf-llist t) ⟨c → t⟩n by simp
    ultimately show ?thesis using proj-active-nth by simp
  qed
  ultimately show ?thesis by simp
qed

lemma baEN1[elim]:
  fixes c::'id
    and t::nat ⇒ cnf
    and t′::nat ⇒ 'cmp
    and n::nat
  assumes act: ∃i. ‖c‖t i
    and nAct: ∄i. i≥n ∧ ‖c‖t i
    and al: eval c t t′ n (ba φ)
  shows φ (t′ (n − ⟨c ∧ t⟩ − 1))
proof −
  from al have φ (lnth ((πc(inf-llist t)) @l (inf-llist t′)) (c↓t(n)))
    using act nAct validCE-cont ba-def by metis
  hence φ (lnth ((πc(inf-llist t)) @l (inf-llist t′)) (c↓t(n))) by simp
  moreover have lnth ((πc(inf-llist t)) @l (inf-llist t′)) (c↓t(n)) = lnth (inf-llist t′) (n − ⟨c ∧ t⟩ − 1)
    using act nAct cnf2bhv-lnth-lappend by simp
  moreover have ... = t′ (n − ⟨c ∧ t⟩ − 1) by simp
  ultimately show ?thesis by simp
qed

lemma baEN2[elim]:
  fixes c::'id
    and t::nat ⇒ cnf
    and t′::nat ⇒ 'cmp
    and n::nat
  assumes nAct: ∄i. ‖c‖t i
    and al: eval c t t′ n (ba φ)
  shows φ (t′ n)
proof −
  from al have φ (lnth ((πc(inf-llist t)) @l (inf-llist t′)) n)
```

using *nAct validCE-not-act ba-def* **by** *metis*
hence φ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*)) *n*) **by** *simp*
moreover have *lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*)) *n* = *lnth* (*inf-llist t'*) *n*
proof −
from *nAct* **have** π_c(*inf-llist t*) = []$_l$ **by** *simp*
hence (π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*) = *inf-llist t'* **by** (*simp add*: ‹π_c*inf-llist t* = []$_l$›)
thus *?thesis* **by** *simp*
qed
moreover have ... = *t' n* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *baEANow*[*elim*]:
fixes *t n c* φ
assumes *eval c t t' n* (*ba* φ)
and $\|c\|_{t\ n}$
shows φ (σ_c(*t n*))
proof −
from *assms* **have** $\varphi(\sigma_c(t\ \langle c \to t\rangle_n))$ **using** *baEA* **by** *blast*
with *assms* **show** *?thesis* **using** *nxtAct-active* **by** *simp*
qed

2.4.9 Next Operator

definition *nxt* :: (*'cmp bta*) ⇒ (*'cmp bta*) ($\bigcirc_b$(-) *24*)
where $\bigcirc_b(\gamma) \equiv \lambda\ t\ n.\ \gamma\ t\ (Suc\ n)$

lemma *nxtIA*[*intro*]:
fixes *c*::*'id*
and *t*::*nat* ⇒ *cnf*
and *t'*::*nat* ⇒ *'cmp*
and *n*::*nat*
assumes $\exists\, i{\geq}n.\ \|c\|_{t\ i}$
and $[\![\exists\, i{>}\langle c \to t\rangle_n.\ \|c\|_{t\ i}]\!] \Longrightarrow \exists\, n' \geq n.\ (\exists!i.\ n{\leq}i \wedge i{<}n' \wedge \|c\|_{t\ i}) \wedge eval\ c\ t\ t'\ n'\ \gamma$
and $[\![\neg(\exists\, i{>}\langle c \to t\rangle_n.\ \|c\|_{t\ i})]\!] \Longrightarrow eval\ c\ t\ t'\ (Suc\ \langle c \to t\rangle_n)\ \gamma$
shows *eval c t t' n* ($\bigcirc_b(\gamma)$)
proof (*cases*)
assume $\exists\, i{>}\langle c \to t\rangle_n.\ \|c\|_{t\ i}$
with *assms*(*2*) **obtain** *n'* **where** *n'*≥*n* **and** $\exists!i.\ n{\leq}i \wedge i{<}n' \wedge \|c\|_{t\ i}$ **and** *eval c t t' n'* γ **by** *blast*
moreover from *assms*(*1*) **have** $\|c\|_{t\ \langle c \to t\rangle_n}$ **and** $\langle c \to t\rangle_n{\geq}n$ **using** *nxtActI* **by** *auto*
ultimately have $\exists\, i'{\geq}n'.\ \|c\|_{t\ i'}$ **by** (*metis* ‹$\exists\, i{>}\langle c \to t\rangle_n.\ \|c\|_{t\ i}$› *dual-order.strict-trans2 leI nat-less-le*)
with ‹*eval c t t' n'* γ›
have γ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*))) (*the-enat* ($\langle c\ \#_{enat\ n'}$ *inf-llist t*$\rangle$))
using *validCE-act* **by** *blast*
moreover have *the-enat*($\langle c\ \#_{n'}$ *inf-llist t*$\rangle$) = *Suc* (*the-enat* ($\langle c\ \#_n$ *inf-llist t*$\rangle$))
proof −
from ‹$\exists!i.\ n{\leq}i \wedge i{<}n' \wedge \|c\|_{t\ i}$› **obtain** *i* **where** *n*≤*i* **and** *i*<*n'* **and** $\|c\|_{t\ i}$
and $\forall\, i'.\ n{\leq}i' \wedge i'{<}n' \wedge \|c\|_{t\ i'} \longrightarrow i'{=}i$ **by** *blast*
moreover have *n' − 1 < llength* (*inf-llist t*) **by** *simp*
ultimately have *the-enat*($\langle c\ \#_{n'}$ *inf-llist t*$\rangle$) = *the-enat*(*eSuc* ($\langle c\ \#_n$ *inf-llist t*$\rangle$))
using *nAct-active-suc*[*of inf-llist t n' n i c*] **by** (*simp add*: ‹*n* ≤ *i*›)
moreover have $\langle c\ \#_i$ *inf-llist t*$\rangle \neq \infty$ **by** *simp*
ultimately show *?thesis* **using** *the-enat-eSuc* **by** *simp*
qed
ultimately have γ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*))) (*Suc* (*the-enat* ($\langle c\ \#_n$ *inf-llist t*$\rangle$)))
by *simp*

with *assms* **have** *eval c t t′ n* (λ*t n*. *γ t* (*Suc n*))
using *validCI-act*[*of n c t* λ*t n*. *γ t* (*Suc n*) *t′*] **by** *blast*
thus *?thesis* **using** *nxt-def* **by** *simp*
next
assume ¬ (∃ *i*>⟨*c* → *t*⟩*n*. ‖*c*‖*t i*)
with *assms*(*3*) **have** *eval c t t′* (*Suc* ⟨*c* → *t*⟩*n*) *γ* **by** *simp*
moreover from ‹¬ (∃ *i*>⟨*c* → *t*⟩*n*. ‖*c*‖*t i*)› **have** ¬ (∃ *i*≥*Suc* ⟨*c* → *t*⟩*n*. ‖*c*‖*t i*) **by** *simp*
ultimately have *γ* (*lnth* (*π*_*c*(*inf-llist t* @*l inf-llist t′*)) (*c*↓*t*(*Suc* ⟨*c* → *t*⟩*n*))
using *assms*(*1*) *validCE-cont*[*of c t Suc* ⟨*c* → *t*⟩*n t′ γ*] **by** *blast*
moreover from *assms*(*1*) ‹¬ (∃ *i*>⟨*c* → *t*⟩*n*. ‖*c*‖*t i*)›
have *Suc* (*the-enat* ⟨*c* #*enat n inf-llist t*⟩) = *c*↓*t*(*Suc* ⟨*c* → *t*⟩*n*)
using *nAct-cnf2proj-Suc-dist* **by** *simp*
ultimately have *γ* (*lnth* ((*π*_*c*(*inf-llist t*)) @*l* (*inf-llist t′*))) (*Suc* (*the-enat* (⟨*c* #*n inf-llist t*⟩)))
by *simp*
moreover from *assms*(*1*) **have** ‖*c*‖*t* ⟨*c* → *t*⟩*n* **and** ⟨*c* → *t*⟩*n* ≥ *n* **using** *nxtActI* **by** *auto*
ultimately have *eval c t t′ n* (λ*t n*. *γ t* (*Suc n*)) **using** *validCI-act*[*of n c t* λ*t n*. *γ t* (*Suc n*) *t′*]
by *blast*
with ‹‖*c*‖*t* ⟨*c* → *t*⟩*n*› ‹¬ (∃ *i′*≥*Suc* ⟨*c* → *t*⟩*n*. ‖*c*‖*t i′*)› **show** *?thesis* **using** *nxt-def* **by** *simp*
qed

lemma *nxtIN*[*intro*]:
fixes *c*::*′id*
and *t*::*nat* ⇒ *cnf*
and *t′*::*nat* ⇒ *′cmp*
and *n*::*nat*
assumes ¬(∃ *i*≥*n*. ‖*c*‖*t i*)
and *eval c t t′* (*Suc n*) *γ*
shows *eval c t t′ n* (○*b*(*γ*))
proof *cases*
assume ∃ *i*. ‖*c*‖*t i*
moreover from ‹¬ (∃ *i*≥*n*. ‖*c*‖*t i*)› **have** ¬ (∃ *i*≥*Suc n*. ‖*c*‖*t i*) **by** *simp*
ultimately have *γ* (*lnth* ((*π*_*c*(*inf-llist t*)) @*l* (*inf-llist t′*))) (*c*↓*t*(*Suc n*))
using *validCE-cont* ‹*eval c t t′* (*Suc n*) *γ*› **by** *blast*
with ‹∃ *i*. ‖*c*‖*t i*› **have** *γ* (*lnth* ((*π*_*c*(*inf-llist t*)) @*l* (*inf-llist t′*))) (*Suc* (*c*↓*t*(*n*)))
using ‹¬(∃ *i*≥*n*. ‖*c*‖*t i*)› *lActive-less* **by** *auto*
with ‹¬(∃ *i*≥*n*. ‖*c*‖*t i*)› ‹∃ *i*. ‖*c*‖*t i*› **have** *eval c t t′ n* (λ*t n*. *γ t* (*Suc n*))
using *validCI-cont*[**where** *γ*=(λ*t n*. *γ t* (*Suc n*))] **by** *simp*
thus *?thesis* **using** *nxt-def* **by** *simp*
next
assume ¬(∃ *i*. ‖*c*‖*t i*)
with *assms* **have** *γ* (*lnth* (*π*_*c*(*inf-llist t* @*l inf-llist t′*)) (*Suc n*) **using** *validCE-not-act* **by** *blast*
with ‹¬(∃ *i*. ‖*c*‖*t i*)› **have** *eval c t t′ n* (λ*t n*. *γ t* (*Suc n*))
using *validCI-not-act*[**where** *γ*=(λ*t n*. *γ t* (*Suc n*))] **by** *blast*
thus *?thesis* **using** *nxt-def* **by** *simp*
qed

lemma *nxtEA1*[*elim*]:
fixes *c*::*′id*
and *t*::*nat* ⇒ *cnf*
and *t′*::*nat* ⇒ *′cmp*
and *n*::*nat*
assumes ∃ *i*>⟨*c* → *t*⟩*n*. ‖*c*‖*t i*
and *eval c t t′ n* (○*b*(*γ*))
and *n′*≥*n*
and ∃!*i*. *i*≥*n* ∧ *i*<*n′* ∧ ‖*c*‖*t i*

shows *eval c t t′ n′ γ*
proof −
from ‹*eval c t t′ n* (○$_b$(*γ*))› **have** *eval c t t′ n* (λ*t n*. *γ t* (*Suc n*)) **using** *nxt-def* **by** *simp*
moreover from *assms*(*4*) **obtain** *i* **where** *i*≥*n* **and** *i*<*n′* **and** ‖*c*‖$_{t\ i}$
and ∀ *i′*. *n*≤*i′* ∧ *i′*<*n′* ∧ ‖*c*‖$_{t\ i'}$ ⟶ *i′*=*i* **by** *blast*
ultimately have *γ* (*lnth* (*π*$_c$*inf-llist t* @$_l$ *inf-llist t′*)) (*Suc* (*the-enat* ⟨*c* #$_{enat\ n}$*inf-llist t*⟩))
using *validCE-act*[*of n c t t′* λ*t n*. *γ t* (*Suc n*)] **by** *blast*
moreover have *the-enat*(⟨*c* #$_{n'}$ *inf-llist t*⟩) = *Suc* (*the-enat* (⟨*c* #$_n$ *inf-llist t*⟩))
proof −
have *n′* − *1* < *llength* (*inf-llist t*) **by** *simp*
with ‹*i*<*n′*› **and** ‹‖*c*‖$_{t\ i}$› **and** ‹∀ *i′*. *n*≤*i′* ∧ *i′*<*n′* ∧ ‖*c*‖$_{t\ i'}$ ⟶ *i′*=*i*›
have *the-enat*(⟨*c* #$_{n'}$ *inf-llist t*⟩) = *the-enat*(*eSuc* (⟨*c* #$_n$ *inf-llist t*⟩))
using *nAct-active-suc*[*of inf-llist t n′ n i c*] **by** (*simp add*: ‹*n* ≤ *i*›)
moreover have ⟨*c* #$_i$ *inf-llist t*⟩ ≠ ∞ **by** *simp*
ultimately show *?thesis* **using** *the-enat-eSuc* **by** *simp*
qed
ultimately have *γ* (*lnth* ((*π*$_c$*inf-llist t*) @$_l$ *inf-llist t′*)) (*the-enat* (⟨*c* #$_{n'}$ *inf-llist t*⟩)) **by** *simp*
moreover have ∃ *i′*≥*n′*. ‖*c*‖$_{t\ i'}$
proof −
from *assms*(*4*) **have** ⟨*c* → *t*⟩$_n$≥*n* **and** ‖*c*‖$_{t\ \langle c\ \to\ t\rangle_n}$ **using** *nxtActI* **by** *auto*
with ‹∀ *i′*. *n*≤*i′* ∧ *i′*<*n′* ∧ ‖*c*‖$_{t\ i'}$ ⟶ *i′*=*i*› **show** *?thesis*
using *assms*(*1*) **by** (*metis leI le-trans less-le*)
qed
ultimately show *?thesis* **using** *validCI-act* **by** *blast*
qed

lemma *nxtEA2*[*elim*]:
fixes *c*::*′id*
and *t*::*nat* ⇒ *cnf*
and *t′*::*nat* ⇒ *′cmp*
and *n*::*nat*
and *i*
assumes ∃ *i*≥*n*. ‖*c*‖$_{t\ i}$ **and** ¬(∃ *i*>⟨*c* → *t*⟩$_n$. ‖*c*‖$_{t\ i}$)
and *eval c t t′ n* (○$_b$(*γ*))
shows *eval c t t′* (*Suc* ⟨*c* → *t*⟩$_n$) *γ*
proof −
from ‹*eval c t t′ n* (○$_b$(*γ*))› **have** *eval c t t′ n* (λ*t n*. *γ t* (*Suc n*)) **using** *nxt-def* **by** *simp*
with *assms*(*1*) **have** *γ* (*lnth* (*π*$_c$*inf-llist t* @$_l$ *inf-llist t′*)) (*Suc* (*the-enat* ⟨*c* #$_{enat\ n}$*inf-llist t*⟩))
using *validCE-act*[*of n c t t′* λ*t n*. *γ t* (*Suc n*)] **by** *blast*
moreover from *assms*(*1*) *assms*(*2*) **have** *Suc* (*the-enat* ⟨*c* #$_{enat\ n}$*inf-llist t*⟩)=$_c$↓$_t$(*Suc* ⟨*c* → *t*⟩$_n$)
using *nAct-cnf2proj-Suc-dist* **by** *simp*
ultimately have *γ* (*lnth* (*π*$_c$*inf-llist t* @$_l$ *inf-llist t′*)) ($_c$↓$_t$(*Suc* ⟨*c* → *t*⟩$_n$)) **by** *simp*
moreover from *assms*(*1*) *assms*(*2*) **have** ¬(∃ *i′*≥*Suc* ⟨*c* → *t*⟩$_n$. ‖*c*‖$_{t\ i'}$)
using *nxtActive-no-active* **by** *simp*
ultimately show *?thesis* **using** *validCI-cont*[**where** *n*=*Suc* ⟨*c* → *t*⟩$_n$] *assms*(*1*) **by** *blast*
qed

lemma *nxtEN*[*elim*]:
fixes *c*::*′id*
and *t*::*nat* ⇒ *cnf*
and *t′*::*nat* ⇒ *′cmp*
and *n*::*nat*
assumes ¬(∃ *i*≥*n*. ‖*c*‖$_{t\ i}$)
and *eval c t t′ n* (○$_b$(*γ*))
shows *eval c t t′* (*Suc n*) *γ*

proof *cases*
assume $\exists\, i.\ \|c\|_{t\ i}$
moreover from ‹*eval c t t′ n* $(\bigcirc_b(\gamma))$› **have** *eval c t t′ n* $(\lambda t\ n.\ \gamma\ t\ (Suc\ n))$ **using** *nxt-def* **by** *simp*
ultimately have γ (*lnth* $(\pi_c$*inf-llist t* $@_l$ *inf-llist t′*)) (*Suc* $({}_c{\downarrow_t}n)$)
using ‹$\neg(\exists\, i{\geq}n.\ \|c\|_{t\ i})$› *validCE-cont*[**where** $\gamma{=}(\lambda t\ n.\ \gamma\ t\ (Suc\ n))$] **by** *simp*
hence γ (*lnth* $((\pi_c$(*inf-llist t*)) $@_l$ (*inf-llist t′*))) $({}_c{\downarrow_t}(Suc\ n))$
using ‹$\exists\, i.\ \|c\|_{t\ i}$› *assms*(*1*) *lActive-less* **by** *auto*
moreover from ‹$\neg\ (\exists\, i{\geq}n.\ \|c\|_{t\ i})$› **have** $\neg\ (\exists\, i{\geq}Suc\ n.\ \|c\|_{t\ i})$ **by** *simp*
ultimately show *?thesis* **using** *validCI-cont*[**where** *n*=*Suc n*] ‹$\exists\, i.\ \|c\|_{t\ i}$› **by** *blast*
next
assume $\neg(\exists\, i.\ \|c\|_{t\ i})$
moreover from ‹*eval c t t′ n* $(\bigcirc_b(\gamma))$› **have** *eval c t t′ n* $(\lambda t\ n.\ \gamma\ t\ (Suc\ n))$ **using** *nxt-def* **by** *simp*
ultimately have γ (*lnth* $(\pi_c$*inf-llist t* $@_l$ *inf-llist t′*)) (*Suc n*)
using ‹$\neg(\exists\, i.\ \|c\|_{t\ i})$› *validCE-not-act*[**where** $\gamma{=}(\lambda t\ n.\ \gamma\ t\ (Suc\ n))$] **by** *blast*
with ‹$\neg(\exists\, i.\ \|c\|_{t\ i})$› **show** *?thesis* **using** *validCI-not-act*[*of c t* γ *t′ Suc n*] **by** *blast*
qed

2.4.10 Eventually Operator

definition *evt* :: (*'cmp bta*) ⇒ (*'cmp bta*) $(\Diamond_b(\text{-})\ 23)$
where $\Diamond_b(\gamma) \equiv \lambda\ t\ n.\ \exists\, n'{\geq}n.\ \gamma\ t\ n'$

lemma *evtIA*[*intro*]:
fixes *c*::*'id*
and *t*::*nat* ⇒ *cnf*
and *t′*::*nat* ⇒ *'cmp*
and *n*::*nat*
and *n′*::*nat*
assumes $\exists\, i{\geq}n.\ \|c\|_{t\ i}$
and $n'{\geq}\langle c \Leftarrow t\rangle_n$
and $[\![\exists\, i{\geq}n'.\ \|c\|_{t\ i}]\!] \Longrightarrow \exists\, n''{\geq}\langle c \Leftarrow t\rangle_{n'}.\ n''{\leq}\ \langle c \rightarrow t\rangle_{n'} \wedge$ *eval c t t′ n″* γ
and $[\![\neg(\exists\, i{\geq}n'.\ \|c\|_{t\ i})]\!] \Longrightarrow$ *eval c t t′ n′* γ
shows *eval c t t′ n* $(\Diamond_b(\gamma))$
proof *cases* **assume** $\exists\, i'{\geq}n'.\ \|c\|_{t\ i'}$
with *assms*(*3*) **obtain** n'' **where** $n'' \geq\langle c \Leftarrow t\rangle_{n'}$ **and** $n''{\leq}\ \langle c \rightarrow t\rangle_{n'}$ **and** *eval c t t′ n″* γ **by** *auto*
hence $\exists\, i'{\geq}n''.\ \|c\|_{t\ i'}$ **using** ‹$\exists\, i'{\geq}n'.\ \|c\|_{t\ i'}$› *nxtActI* **by** *blast*
with ‹*eval c t t′ n″* γ› **have**
γ (*lnth* $((\pi_c$(*inf-llist t*)) $@_l$ (*inf-llist t′*))) (*the-enat* $(\langle c\ \#_{n''}$ *inf-llist t*$\rangle)$)
using *validCE-act* **by** *blast*
moreover have *the-enat* $(\langle c\ \#_n$ *inf-llist t*$\rangle) \leq$ *the-enat* $(\langle c\ \#_{n''}$ *inf-llist t*$\rangle)$
proof −
from ‹$\langle c \Leftarrow t\rangle_{n'}{\leq}n''$› **have** $\langle c\ \#_{n'}$ *inf-llist t*$\rangle \leq \langle c\ \#_{n''}$ *inf-llist t*$\rangle$
using *nAct-mono-lNact* **by** *simp*
moreover from ‹$n'{\geq}\langle c \Leftarrow t\rangle_n$› **have** $\langle c\ \#_n$ *inf-llist t*$\rangle \leq \langle c\ \#_{n'}$ *inf-llist t*$\rangle$
using *nAct-mono-lNact* **by** *simp*
moreover have $\langle c\ \#_{n'}$ *inf-llist t*$\rangle \neq \infty$ **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
moreover have $\exists\, i'{\geq}n.\ \|c\|_{t\ i'}$
proof −
from ‹$\exists\, i'{\geq}n'.\ \|c\|_{t\ i'}$› **obtain** *i′* **where** $i'{\geq}n'$ **and** $\|c\|_{t\ i'}$ **by** *blast*
with ‹$n'{\geq}\langle c \Leftarrow t\rangle_n$› **have** $i'{\geq}\ n$ **using** *lNactGe le-trans* **by** *blast*
with ‹$\|c\|_{t\ i'}$› **show** *?thesis* **by** *blast*
qed
ultimately have *eval c t t′ n* $(\lambda t\ n.\ \exists\, n'{\geq}n.\ \gamma\ t\ n')$
using *validCI-act*[**where** $\gamma{=}(\lambda t\ n.\ \exists\, n'{\geq}n.\ \gamma\ t\ n')$] **by** *blast*

thus *?thesis* **using** *evt-def* **by** *simp*
next
assume $\neg(\exists\, i'{\geq}n'.\ \|c\|_{t\ i'})$
with $\langle(\exists\, i{\geq}n.\ \|c\|_{t\ i})\rangle$ **have** $n' \geq \langle c \wedge t\rangle$ **using** *lActive-less* **by** *auto*
hence $_{c{\downarrow}t}(n') \geq$ *the-enat* (*llength* (π_c(*inf-llist t*))) − *1* **using** *cnf2bhv-ge-llength* **by** *simp*
moreover have *the-enat*(*llength* (π_c(*inf-llist t*))) − *1* ≥ *the-enat*($\langle c\ \#_n$ *inf-llist t*$\rangle$)
proof −
from $\langle\exists\, i{\geq}n.\ \|c\|_{t\ i}\rangle$ **have** *llength* (π_c(*inf-llist t*)) ≥ *eSuc* ($\langle c\ \#_n$ *inf-llist t*$\rangle$)
using *nAct-llength-proj* **by** *simp*
moreover from $\langle\neg(\exists\, i'{\geq}n'.\ \|c\|_{t\ i'})\rangle$ **have** *lfinite* (π_c(*inf-llist t*))
using *proj-finite2*[*of inf-llist t*] **by** *simp*
hence *llength* (π_c(*inf-llist t*))≠∞ **using** *llength-eq-infty-conv-lfinite* **by** *auto*
ultimately have *the-enat* (*llength* (π_c(*inf-llist t*))) ≥ *the-enat*(*eSuc* ($\langle c\ \#_n$ *inf-llist t*$\rangle$))
by *simp*
moreover have $\langle c\ \#_n$ *inf-llist t*$\rangle$≠∞ **by** *simp*
ultimately have *the-enat* (*llength* (π_c(*inf-llist t*))) ≥ *Suc* (*the-enat* ($\langle c\ \#_n$ *inf-llist t*$\rangle$))
using *the-enat-eSuc* **by** *simp*
thus *?thesis* **by** *simp*
qed
ultimately have $_{c{\downarrow}t}(n') \geq$ *the-enat* ($\langle c\ \#_n$ *inf-llist t*$\rangle$) **by** *simp*
moreover from $\langle\neg(\exists\, i'{\geq}n'.\ \|c\|_{t\ i'})\rangle$ **have** *eval c t t′ n′ γ* **using** *assms*(*4*) **by** *simp*
with $\langle\exists\, i{\geq}n.\ \|c\|_{t\ i}\rangle$ $\langle\neg(\exists\, i'{\geq}n'.\ \|c\|_{t\ i'})\rangle$
have *γ* (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t′*))) ($_{c{\downarrow}t}(n')$) **using** *validCE-cont* **by** *blast*
ultimately have *eval c t t′ n* (*λt n.* ∃ *n′*≥*n.* *γ t n′*)
using $\langle\exists\, i{\geq}n.\ \|c\|_{t\ i}\rangle$ *validCI-act*[**where** *γ*=(*λt n.* ∃ *n′*≥*n.* *γ t n′*)] **by** *blast*
thus *?thesis* **using** *evt-def* **by** *simp*
qed

lemma *evtIN*[*intro*]:
fixes *c*::*′id*
and *t*::*nat* ⇒ *cnf*
and *t′*::*nat* ⇒ *′cmp*
and *n*::*nat*
and *n′*::*nat*
assumes ¬(∃ *i*≥*n.* $\|c\|_{t\ i}$)
and *n′*≥*n*
and *eval c t t′ n′ γ*
shows *eval c t t′ n* (◇$_b$(*γ*))
proof *cases*
assume ∃ *i.* $\|c\|_{t\ i}$
moreover from *assms*(*1*) *assms*(*2*) **have** ¬(∃ *i′*≥*n′.* $\|c\|_{t\ i'}$) **by** *simp*
ultimately have *γ* (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t′*))) ($_{c{\downarrow}t}(n')$)
using *validCE-cont*[*of c t n′ t′ γ*] ⟨*eval c t t′ n′ γ*⟩ **by** *blast*
moreover from ⟨*n′*≥*n*⟩ **have** $_{c{\downarrow}t}(n') \geq$ $_{c{\downarrow}t}(n)$ **using** *cnf2bhv-mono* **by** *simp*
ultimately have *eval c t t′ n* (*λt n.* ∃ *n′*≥*n.* *γ t n′*)
using *validCI-cont*[**where** *γ*=(*λt n.* ∃ *n′*≥*n.* *γ t n′*)] ⟨∃ *i.* $\|c\|_{t\ i}$⟩ ⟨¬(∃ *i*≥*n.* $\|c\|_{t\ i}$)⟩ **by** *blast*
thus *?thesis* **using** *evt-def* **by** *simp*
next
assume ¬(∃ *i.* $\|c\|_{t\ i}$)
with *assms* **have** *γ* (*lnth* (π_c*inf-llist t* @$_l$ *inf-llist t′*)) *n′* **using** *validCE-not-act* **by** *blast*
with ⟨¬(∃ *i.* $\|c\|_{t\ i}$)⟩ **have** *eval c t t′ n* (*λt n.* ∃ *n′*≥*n.* *γ t n′*)
using *validCI-not-act*[**where** *γ*=*λt n.* ∃ *n′*≥*n.* *γ t n′*] ⟨*n′*≥*n*⟩ **by** *blast*
thus *?thesis* **using** *evt-def* **by** *simp*
qed

lemma *evtEA*[*elim*]:
fixes c::*'id*
and t::*nat* $\Rightarrow$ *cnf*
and t'::*nat* $\Rightarrow$ *'cmp*
and n::*nat*
assumes $\exists\, i{\geq}n.\ \|c\|_{t\ i}$
and *eval c t t' n* $(\Diamond_b(\gamma))$
shows $\exists\, n'{\geq}\langle c \to t\rangle_n$.
$(\exists\, i{\geq}n'.\ \|c\|_{t\ i} \wedge (\forall\, n''{\geq}\ \langle c \Leftarrow t\rangle_{n'}.\ n''{\leq}\langle c \to t\rangle_{n'} \longrightarrow \mathit{eval}\ c\ t\ t'\ n''\ \gamma)) \vee$
$(\neg(\exists\, i{\geq}n'.\ \|c\|_{t\ i}) \wedge \mathit{eval}\ c\ t\ t'\ n'\ \gamma)$
proof −
from ‹*eval c t t' n* $(\Diamond_b(\gamma))$› **have** *eval c t t' n* $(\lambda t\ n.\ \exists\, n'{\geq}n.\ \gamma\ t\ n')$ **using** *evt-def* **by** *simp*
with ‹$\exists\, i{\geq}n.\ \|c\|_{t\ i}$›
have $\exists\, n'{\geq}$*the-enat* $\langle c\ \#_{\mathit{enat}\ n}$*inf-llist t*$\rangle$. γ (*lnth* (π_c*inf-llist t* $@_l$ *inf-llist t'*)) n'
using *validCE-act*[**where** $\gamma{=}\lambda t\ n.\ \exists\, n'{\geq}n.\ \gamma\ t\ n'$] **by** *blast*
then obtain x **where** $x{\geq}$*the-enat* ($\langle c\ \#_n$ *inf-llist t*$\rangle$) **and**
γ (*lnth* ((π_c(*inf-llist t*)) $@_l$ (*inf-llist t'*))) x **by** *auto*
thus *?thesis*
proof (*cases*)
assume $x \geq$ *llength* (π_c(*inf-llist t*))
moreover from ‹($x \geq$ *llength* (π_c(*inf-llist t*)))› **have** *llength* (π_c(*inf-llist t*))$\neq\infty$
by (*metis infinity-ileE*)
moreover from ‹$\exists\, i{\geq}n.\ \|c\|_{t\ i}$› **have** *llength* ($\pi_c$(*inf-llist t*))$\geq$*1*
using *proj-one*[*of inf-llist t*] **by** *auto*
ultimately have *the-enat* (*llength* (π_c(*inf-llist t*))) − *1* $<$ x
by (*metis One-nat-def Suc-ile-eq antisym-conv2 diff-Suc-less enat-ord-simps*(*2*)
enat-the-enat less-imp-diff-less one-enat-def)
hence $x = {c\downarrow_t}(c{\uparrow_t}(x))$ **using** *cnf2bhv-bhv2cnf* **by** *simp*
with ‹γ (*lnth* ((π_c(*inf-llist t*)) $@_l$ (*inf-llist t'*))) x›
have γ (*lnth* ((π_c(*inf-llist t*)) $@_l$ (*inf-llist t'*))) (${c\downarrow_t}(c{\uparrow_t}(x))$) **by** *simp*
moreover have $\neg(\exists\, i{\geq}c{\uparrow_t}(x).\ \|c\|_{t\ i})$
proof −
from ‹$x \geq$ *llength* (π_c(*inf-llist t*))› **have** *lfinite* (π_c(*inf-llist t*))
using *llength-geq-enat-lfiniteD*[*of* π_c(*inf-llist t*) x] **by** *simp*
then obtain z **where** $\forall\, n''{>}z.\ \neg\ \|c\|_{t\ n''}$ **using** *proj-finite-bound* **by** *blast*
moreover from ‹*the-enat* (*llength* (π_c(*inf-llist t*))) − *1* $<$ x› **have** $\langle c \wedge t\rangle < c{\uparrow_t}(x)$
using *bhv2cnf-greater-lActive* **by** *simp*
ultimately show *?thesis* **using** *lActive-greater-active-all* **by** *simp*
qed
ultimately have *eval c t t'* $(c{\uparrow_t}x)$ γ
using ‹$\exists\, i{\geq}n.\ \|c\|_{t\ i}$› *validCI-cont*[*of c t* $c{\uparrow_t}(x)$] **by** *blast*
moreover have $c{\uparrow_t}(x) \geq \langle c \to t\rangle_n$
proof −
from ‹$x \geq$ *llength* (π_c(*inf-llist t*))› **have** *lfinite* (π_c(*inf-llist t*))
using *llength-geq-enat-lfiniteD*[*of* π_c(*inf-llist t*) x] **by** *simp*
then obtain z **where** $\forall\, n''{>}z.\ \neg\ \|c\|_{t\ n''}$ **using** *proj-finite-bound* **by** *blast*
moreover from ‹$\exists\, i{\geq}n.\ \|c\|_{t\ i}$› **have** $\|c\|_{t\ \langle c \to t\rangle_n}$ **using** *nxtActI* **by** *simp*
ultimately have $\langle c \wedge t\rangle{\geq}\langle c \to t\rangle_n$ **using** *lActive-greatest* **by** *fastforce*
moreover have $c{\uparrow_t}(x) \geq \langle c \wedge t\rangle$ **by** *simp*
ultimately show $c{\uparrow_t}(x) \geq \langle c \to t\rangle_n$ **by** *arith*
qed
ultimately show *?thesis* **using** ‹$\neg(\exists\, i{\geq}c{\uparrow_t}(x).\ \|c\|_{t\ i})$› **by** *blast*
next
assume $\neg(x \geq$ *llength* (π_c(*inf-llist t*)))
hence $x{<}$*llength* (π_c(*inf-llist t*)) **by** *simp*

```
    then obtain n'::nat where x=⟨c #_n' inf-llist t⟩ using nAct-exists by blast
    with ‹enat x < llength (π_c(inf-llist t))› have ∃ i≥n'. ‖c‖_t i using nAct-less-llength-active by force
    then obtain i where i≥n' and ‖c‖_t i and ¬ (∃ k≥n'. k < i ∧ ‖c‖_t k) using nact-exists by blast
    moreover have (∀ n''≥ ⟨c ⇐ t⟩_i. n''≤⟨c → t⟩_i ⟶ eval c t t' n'' γ)
    proof
      fix n'' show ⟨c ⇐ t⟩_i ≤ n'' ⟶ n'' ≤ ⟨c → t⟩_i ⟶ eval c t t' n'' γ
      proof(rule HOL.impI[OF HOL.impI])
        assume ⟨c ⇐ t⟩_i ≤ n'' and n'' ≤ ⟨c → t⟩_i
        hence the-enat (⟨c #_enat i inf-llist t⟩) = the-enat (⟨c #_enat n'' inf-llist t⟩)
          using nAct-same by simp
        moreover from ‹‖c‖_t i› have ‖c‖_t ⟨c → t⟩_i using nxtActI by auto
        with ‹n'' ≤ ⟨c → t⟩_i› have ∃ i≥n''. ‖c‖_t i using dual-order.strict-implies-order by auto
        moreover have γ (lnth ((π_c(inf-llist t)) @_l (inf-llist t'))) (the-enat (⟨c #_enat i inf-llist t⟩))
        proof −
          have enat i − 1 < llength (inf-llist t) by (simp add: one-enat-def)
          with ‹x=⟨c #_n' inf-llist t⟩› ‹i≥n'› ‹¬ (∃ k≥n'. k < i ∧ ‖c‖_t k)› have x=⟨c #_i inf-llist t⟩
            using one-enat-def nAct-not-active-same by simp
          moreover have ⟨c #_i inf-llist t⟩≠∞ by simp
          ultimately have x=the-enat(⟨c #_i inf-llist t⟩) by fastforce
          thus ?thesis using ‹γ (lnth ((π_c(inf-llist t)) @_l (inf-llist t'))) x› by blast
        qed
        with ‹the-enat (⟨c #_enat i inf-llist t⟩) = the-enat (⟨c #_enat n'' inf-llist t⟩)› have
          γ (lnth ((π_c(inf-llist t)) @_l (inf-llist t'))) (the-enat (⟨c #_enat n'' inf-llist t⟩)) by simp
        ultimately show eval c t t' n'' γ using validCI-act by blast
      qed
    qed
    moreover have i≥⟨c → t⟩_n
    proof −
      have enat i − 1 < llength (inf-llist t) by (simp add: one-enat-def)
      with ‹x=⟨c #_n' inf-llist t⟩› ‹i≥n'› ‹¬ (∃ k≥n'. k < i ∧ ‖c‖_t k)› have x=⟨c #_i inf-llist t⟩
        using one-enat-def nAct-not-active-same by simp
      moreover have ⟨c #_i inf-llist t⟩≠∞ by simp
      ultimately have x=the-enat(⟨c #_i inf-llist t⟩) by fastforce
      with ‹x≥the-enat (⟨c #_n inf-llist t⟩)›
        have the-enat (⟨c #_i inf-llist t⟩)≥the-enat (⟨c #_n inf-llist t⟩) by simp
      with ‹‖c‖_t i› show ?thesis using active-geq-nxtAct by simp
    qed
    ultimately show ?thesis using ‹‖c‖_t i› by auto
  qed
qed

lemma evtEN[elim]:
  fixes c::'id
    and t::nat ⇒ cnf
    and t'::nat ⇒ 'cmp
    and n::nat
    and n'::nat
  assumes ¬(∃ i≥n. ‖c‖_t i)
    and eval c t t' n (◇_b(γ))
  shows ∃ n'≥n. eval c t t' n' γ
proof cases
  assume ∃ i. ‖c‖_t i
  moreover from ‹eval c t t' n (◇_b(γ))› have eval c t t' n (λt n. ∃ n'≥n. γ t n') using evt-def by simp
  ultimately have ∃ n'≥_c↓_t n. γ (lnth (π_c inf-llist t @_l inf-llist t')) n'
    using validCE-cont[where γ=(λt n. ∃ n'≥n. γ t n')] ‹¬(∃ i≥n. ‖c‖_t i)› by blast
```

then obtain x **where** $x \geq_c \downarrow_t(n)$ **and** $\gamma\ (lnth\ ((\pi_c(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t'))\ x$ **by** *auto*
moreover have *the-enat* $(llength\ (\pi_c(inf\text{-}llist\ t))) - 1 < x$
proof −
have $\langle c \wedge t \rangle < n$
proof (*rule ccontr*)
assume $\neg\langle c \wedge t \rangle < n$
hence $\langle c \wedge t \rangle \geq n$ **by** *simp*
moreover from ‹$\exists\, i.\ \|c\|_{t\ i}$› ‹$\neg\ (\exists\, i \geq n.\ \|c\|_{t\ i})$› **have** $\|c\|_{t\ \langle c \wedge t \rangle}$
using *lActive-active less-or-eq-imp-le* **by** *blast*
ultimately show *False* **using** ‹$\neg\ (\exists\, i \geq n.\ \|c\|_{t\ i})$› **by** *simp*
qed
hence *the-enat* $(llength\ (\pi_c(inf\text{-}llist\ t))) - 1 <\ _c\downarrow_t(n)$ **using** *cnf2bhv-greater-llength* **by** *simp*
with ‹$x \geq_c \downarrow_t(n)$› **show** *?thesis* **by** *simp*
qed
hence $x =\ _c\downarrow_t(_c\uparrow_t(x))$ **using** *cnf2bhv-bhv2cnf* **by** *simp*
ultimately have $\gamma\ (lnth\ ((\pi_c(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ (_c\downarrow_t(_c\uparrow_t(x)))$ **by** *simp*
moreover from ‹$\neg(\exists\, i \geq n.\ \|c\|_{t\ i})$› **have** $\neg(\exists\, i \geq\ _c\uparrow_t(x).\ \|c\|_{t\ i})$
proof −
from ‹$\neg(\exists\, i \geq n.\ \|c\|_{t\ i})$› **have** *lfinite* $(\pi_c(inf\text{-}llist\ t))$ **using** *proj-finite2* **by** *simp*
then obtain z **where** $\forall\, n'' > z.\ \neg\ \|c\|_{t\ n''}$ **using** *proj-finite-bound* **by** *blast*
moreover from ‹*the-enat* $(llength\ (\pi_c(inf\text{-}llist\ t))) - 1 < x$› **have** $\langle c \wedge t \rangle <\ _c\uparrow_t(x)$
using *bhv2cnf-greater-lActive* **by** *simp*
ultimately show *?thesis* **using** *lActive-greater-active-all* **by** *simp*
qed
ultimately have *eval c t t'* $(_c\uparrow_t x)\ \gamma$
using *validCI-cont*[*of c t* $_c\uparrow_t(x)\ \gamma$] ‹$\exists\, i.\ \|c\|_{t\ i}$› **by** *blast*
moreover from ‹$\exists\, i.\ \|c\|_{t\ i}$› ‹$\neg(\exists\, i \geq n.\ \|c\|_{t\ i})$› **have** $\langle c \wedge t \rangle \leq n$ **using** *lActive-less*[*of c t - n*] **by** *auto*
with ‹$x \geq_c \downarrow_t(n)$› **have** $n \leq\ _c\uparrow_t(x)$ **using** *p2c-mono-c2p* **by** *blast*
ultimately show *?thesis* **by** *auto*
next
assume $\neg(\exists\, i.\ \|c\|_{t\ i})$
moreover from ‹*eval c t t' n* $(\Diamond_b(\gamma))$› **have** *eval c t t' n* $(\lambda t\ n.\ \exists\, n' \geq n.\ \gamma\ t\ n')$ **using** *evt-def* **by** *simp*
ultimately obtain n' **where** $n' \geq n$ **and** $\gamma\ (lnth\ (\pi_c inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ n'$
using ‹$\neg(\exists\, i.\ \|c\|_{t\ i})$› *validCE-not-act*[**where** $\gamma = \lambda t\ n.\ \exists\, n' \geq n.\ \gamma\ t\ n'$] **by** *blast*
with ‹$\neg(\exists\, i.\ \|c\|_{t\ i})$› **show** *?thesis* **using** *validCI-not-act*[*of c t* γ *t' n'*] **by** *blast*
qed

2.4.11 Globally Operator

definition *glob* :: (*'cmp bta*) ⇒ (*'cmp bta*) ($\Box_b$(-) *22*)
where $\Box_b(\gamma) \equiv \lambda\ t\ n.\ \forall\, n' \geq n.\ \gamma\ t\ n'$

lemma *globIA*[*intro*]:
fixes *c*::*'id*
and *t*::*nat* ⇒ *cnf*
and *t'*::*nat* ⇒ *'cmp*
and *n*::*nat*
assumes $\exists\, i \geq n.\ \|c\|_{t\ i}$
and $\bigwedge n'.\ [\![\exists\, i \geq n'.\ \|c\|_{t\ i};\ n' \geq \langle c \rightarrow t \rangle_n]\!] \Longrightarrow \exists\, n'' \geq \langle c \Leftarrow t \rangle_{n'}.\ n'' \leq \langle c \rightarrow t \rangle_{n'} \wedge$ *eval c t t' n''* γ
and $\bigwedge n'.\ [\![\neg(\exists\, i \geq n'.\ \|c\|_{t\ i});\ n' \geq \langle c \rightarrow t \rangle_n]\!] \Longrightarrow$ *eval c t t' n'* γ
shows *eval c t t' n* $(\Box_b(\gamma))$
proof −
have $\forall\, n' \geq$ *the-enat* $\langle c\ \#_{enat\ n} inf\text{-}llist\ t \rangle.\ \gamma\ (lnth\ (\pi_c inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ n'$
proof
fix *x*::*nat* **show**

$x \geq$*the-enat* $(\langle c \#_n$ *inf-llist* $t\rangle) \longrightarrow \gamma$ $(lnth$ $(\pi_c$*inf-llist* t $@_l$ *inf-llist* $t'))$ x

proof

assume $x \geq$*the-enat* $(\langle c \#_n$ *inf-llist* $t\rangle)$

show γ $(lnth$ $((\pi_c($*inf-llist* $t))$ $@_l$ $($*inf-llist* $t')))$ x

proof (*cases*)

assume $(x \geq$ *llength* $(\pi_c($*inf-llist* $t)))$

hence *lfinite* $(\pi_c($*inf-llist* $t))$

using *llength-geq-enat-lfiniteD*[*of* $\pi_c($*inf-llist* $t)$ x] **by** *simp*

then obtain z **where** $\forall\, n''{>}z.\ \neg\, \|c\|_{t\ n''}$ **using** *proj-finite-bound* **by** *blast*

moreover have $\|c\|_{t\ \langle c \to t\rangle_n}$ **by** (*simp add*: $\langle \exists\, i{\geq}n.\ \|c\|_{t\ i}\rangle$ *nxtActI*)

ultimately have $\langle c \wedge t\rangle \geq \langle c \to t\rangle_n$ **using** *lActive-greatest*[*of* c t $\langle c \to t\rangle_n$] **by** *blast*

moreover have ${}_c{\uparrow}_t(x) \geq \langle c \wedge t\rangle$ **by** *simp*

ultimately have ${}_c{\uparrow}_t(x) \geq \langle c \to t\rangle_n$ **by** *arith*

moreover have $\neg\, (\exists\, i'{\geq}{}_c{\uparrow}_t(x).\ \|c\|_{t\ i'})$

proof −

from $\langle$*lfinite* $(\pi_c($*inf-llist* $t))\rangle$ $\langle \exists\, i{\geq}n.\ \|c\|_{t\ i}\rangle$

have ${}_c{\uparrow}_t($*the-enat* $($*llength* $(\pi_c($*inf-llist* $t))))) =$ *Suc* $(\langle c \wedge t\rangle)$

using *bhv2cnf-lActive* **by** *blast*

moreover from $\langle (x \geq$ *llength* $(\pi_c($*inf-llist* $t)))\rangle$ **have** $x \geq$ *the-enat*$($*llength* $(\pi_c($*inf-llist* $t)))$

using *the-enat-mono* **by** *fastforce*

hence ${}_c{\uparrow}_t(x) \geq {}_c{\uparrow}_t($*the-enat* $($*llength* $(\pi_c($*inf-llist* $t))))$

using *bhv2cnf-mono*[*of the-enat* $($*llength* $(\pi_c($*inf-llist* $t))))$ x] **by** *simp*

ultimately have ${}_c{\uparrow}_t(x) \geq$ *Suc* $(\langle c \wedge t\rangle)$ **by** *simp*

hence ${}_c{\uparrow}_t(x) > \langle c \wedge t\rangle$ **by** *simp*

with $\langle \forall\, n''{>}z.\ \neg\, \|c\|_{t\ n''}\rangle$ **show** *?thesis* **using** *lActive-greater-active-all* **by** *simp*

qed

ultimately have *eval* c t t' $({}_c{\uparrow}_t(x))$ γ **using** *assms*(*3*) **by** *simp*

hence γ $(lnth$ $((\pi_c($*inf-llist* $t))$ $@_l$ $($*inf-llist* $t')))$ $({}_c{\downarrow}_t({}_c{\uparrow}_t(x)))$

using *validCE-cont*[*of* c t ${}_c{\uparrow}_t(x)$ t' γ] $\langle \exists\, i{\geq}n.\ \|c\|_{t\ i}\rangle$ $\langle \neg\ (\exists\, i'{\geq}{}_c{\uparrow}_t(x).\ \|c\|_{t\ i'})\rangle$ **by** *blast*

moreover from $\langle (x \geq$ *llength* $(\pi_c($*inf-llist* $t)))\rangle$

have $($*enat* $x \geq$ *llength* $(\pi_c($*inf-llist* $t)))$ **by** *auto*

with $\langle$*lfinite* $(\pi_c($*inf-llist* $t))\rangle$ **have** *llength* $(\pi_c($*inf-llist* $t)) \neq \infty$

using *llength-eq-infty-conv-lfinite* **by** *auto*

with $\langle (x \geq$ *llength* $(\pi_c($*inf-llist* $t)))\rangle$

have *the-enat*$($*llength* $(\pi_c($*inf-llist* $t))) - 1 \leq x$ **by** *auto*

ultimately show *?thesis* **using** *cnf2bhv-bhv2cnf*[*of* c t x] **by** *simp*

next

assume $\neg(x \geq$ *llength* $(\pi_c($*inf-llist* $t)))$

hence $x <$ *llength* $(\pi_c($*inf-llist* $t))$ **by** *simp*

then obtain n'::*nat* **where** $x = \langle c \#_{n'}$ *inf-llist* $t\rangle$ **using** *nAct-exists* **by** *blast*

moreover from $\langle$*enat* $x <$ *llength* $(\pi_c($*inf-llist* $t))\rangle$ $\langle$*enat* $x = \langle c \#_{enat\ n'}$ *inf-llist* $t\rangle\rangle$

have $\exists\, i{\geq}n'.\ \|c\|_{t\ i}$ **using** *nAct-less-llength-active* **by** *force*

then obtain i **where** $i{\geq}n'$ **and** $\|c\|_{t\ i}$ **and** $\neg\ (\exists\, k{\geq}n'.\ k < i \wedge \|c\|_{t\ k})$

using *nact-exists* **by** *blast*

moreover have *enat* $i - 1 <$ *llength* $($*inf-llist* $t)$ **by** (*simp add*: *one-enat-def*)

ultimately have $x = \langle c \#_i$ *inf-llist* $t\rangle$ **using** *one-enat-def nAct-not-active-same* **by** *simp*

moreover have $\langle c \#_i$ *inf-llist* $t\rangle \neq \infty$ **by** *simp*

ultimately have $x =$*the-enat*$(\langle c \#_i$ *inf-llist* $t\rangle)$ **by** *fastforce*

from $\langle x \geq$*the-enat* $(\langle c \#_n$ *inf-llist* $t\rangle)\rangle$ $\langle x =$*the-enat*$(\langle c \#_i$ *inf-llist* $t\rangle)\rangle$

have *the-enat* $(\langle c \#_i$ *inf-llist* $t\rangle) \geq$*the-enat* $(\langle c \#_n$ *inf-llist* $t\rangle)$ **by** *simp*

with $\langle \|c\|_{t\ i}\rangle$ **have** $i \geq \langle c \to t\rangle_n$ **using** *active-geq-nxtAct* **by** *simp*

moreover from $\langle x = \langle c \#_i$ *inf-llist* $t\rangle\rangle$ $\langle x <$ *llength* $(\pi_c($*inf-llist* $t))\rangle$

have $\exists\, i'.\ i \leq$ *enat* $i' \wedge \|c\|_{t\ i'}$ **using** *nAct-less-llength-active*[*of* x c *inf-llist* t i] **by** *simp*

hence $\exists\, i'{\geq}i.\ \|c\|_{t\ i'}$ **by** *simp*

ultimately obtain n'' **where** *eval* c t t' n'' γ **and** $n'' \geq \langle c \Leftarrow t\rangle_i$ **and** $n'' \leq \langle c \to t\rangle_i$

```
      using assms(2) by blast
    moreover have ∃ i'≥n''. ‖c‖_{t i'}
      using ‹‖c‖_{t i}› ‹n''≤⟨c → t⟩_i› less-or-eq-imp-le nxtAct-active by auto
    ultimately have γ (lnth ((π_c(inf-llist t)) @_l (inf-llist t'))) (the-enat (⟨c #_{n''} inf-llist t⟩))
      using validCE-act[of n'' c t t' γ] by blast
    moreover from ‹n''≥⟨c ⇐ t⟩_i› and ‹n''≤⟨c → t⟩_i›
      have the-enat (⟨c #_{n''} inf-llist t⟩)=the-enat (⟨c #_i inf-llist t⟩) using nAct-same by simp
    hence the-enat (⟨c #_{n''} inf-llist t⟩) = x by (simp add: ‹x = the-enat ⟨c #_{enat i} inf-llist t⟩›)
    ultimately have γ (lnth ((π_c(inf-llist t)) @_l (inf-llist t'))) (the-enat x) by simp
    thus ?thesis by simp
  qed
 qed
 qed
 with ‹∃ i≥n. ‖c‖_{t i}› have eval c t t' n (λt n. ∀ n'≥n. γ t n')
  using validCI-act[of n c t λ t n. ∀ n'≥n. γ t n' t'] by blast
 thus ?thesis using glob-def by simp
qed

lemma globIN[intro]:
 fixes c::'id
  and t::nat ⇒ cnf
  and t'::nat ⇒ 'cmp
  and n::nat
 assumes ¬(∃ i≥n. ‖c‖_{t i})
  and ⋀n'. n'≥n ⟹ eval c t t' n' γ
 shows eval c t t' n (□_b(γ))
proof cases
 assume ∃ i. ‖c‖_{t i}
 from ‹¬(∃ i≥n. ‖c‖_{t i})› have lfinite (π_c(inf-llist t)) using proj-finite2 by simp
 then obtain z where ∀ n''>z. ¬ ‖c‖_{t n''} using proj-finite-bound by blast

 have ∀ x::nat≥ c↓_t(n). γ (lnth (π_c inf-llist t @_l inf-llist t')) x
 proof
  fix x::nat show (x≥c↓_t(n)) ⟶ γ (lnth (π_c inf-llist t @_l inf-llist t')) x
  proof
   assume x≥c↓_t(n)
   moreover from ‹¬(∃ i≥n. ‖c‖_{t i})› have ⟨c ∧ t⟩ ≤ n using ‹∃ i. ‖c‖_{t i}› lActive-less by auto
   ultimately have c↑_t(x) ≥ n using p2c-mono-c2p by simp
   with assms have eval c t t' (c↑_t(x)) γ by simp
   moreover have ¬ (∃ i'≥c↑_t(x). ‖c‖_{t i'})
   proof −
    from ‹lfinite (π_c(inf-llist t))› ‹∃ i. ‖c‖_{t i}›
     have c↑_t(the-enat (llength (π_c(inf-llist t)))) = Suc (⟨c ∧ t⟩)
     using bhv2cnf-lActive by blast
    moreover from ‹¬(∃ i≥n. ‖c‖_{t i})› have n>⟨c ∧ t⟩
     by (meson ‹∃ i. ‖c‖_{t i}› lActive-active leI le-eq-less-or-eq)
    hence n≥Suc (⟨c ∧ t⟩) by simp
    with ‹n≥Suc(⟨c ∧ t⟩)› ‹c↑_t(x) ≥ n› have c↑_t(x) ≥ Suc (⟨c ∧ t⟩) by simp
    hence c↑_t(x) > ⟨c ∧ t⟩ by simp
    with ‹∀ n''>z. ¬ ‖c‖_{t n''}› show ?thesis using lActive-greater-active-all by simp
   qed
   ultimately have γ (lnth ((π_c(inf-llist t)) @_l (inf-llist t'))) (c↓_t(c↑_t(x)))
    using validCE-cont[of c t c↑_t(x) t' γ] ‹∃ i. ‖c‖_{t i}› by blast
   moreover have x ≥ the-enat (llength (π_c(inf-llist t))) − 1
    using ‹c↓_t(n) ≤ x› cnf2bhv-def by auto
```

```
    ultimately show γ (lnth ((πc(inf-llist t)) @l (inf-llist t′))) x
      using cnf2bhv-bhv2cnf by simp
  qed
 qed
 with ‹∃ i. ‖c‖t i› ‹¬(∃ i≥n. ‖c‖t i)› have eval c t t′ n (λt n. ∀ n′≥n. γ t n′)
  using validCI-cont[of c t n λ t n. ∀ n′≥n. γ t n′ t′] by simp
 thus ?thesis using glob-def by simp
next
 assume ¬(∃ i. ‖c‖t i)
 with assms have ∀ n′≥n. γ (lnth (πcinf-llist t @l inf-llist t′)) n′ using validCE-not-act by blast
 with ‹¬(∃ i. ‖c‖t i)› have eval c t t′ n (λt n. ∀ n′≥n. γ t n′)
  using validCI-not-act[where γ=λ t n. ∀ n′≥n. γ t n′] by blast
 thus ?thesis using glob-def by simp
qed

lemma globEA[elim]:
 fixes c::′id
  and t::nat ⇒ cnf
  and t′::nat ⇒ ′cmp
  and n::nat
  and n′::nat
 assumes ∃ i≥n. ‖c‖t i
  and eval c t t′ n (□b(γ))
  and n′≥⟨c ⇐ t⟩n
 shows eval c t t′ n′ γ
proof (cases)
 assume ∃ i≥n′. ‖c‖t i
 with ‹n′≥⟨c ⇐ t⟩n› have the-enat (⟨c #n′ inf-llist t⟩) ≥ the-enat (⟨c #n inf-llist t⟩)
  using nAct-mono-lNact ‹∃ i≥n. ‖c‖t i› by simp
 moreover from ‹eval c t t′ n (□b(γ))› have eval c t t′ n (λt n. ∀ n′≥n. γ t n′)
  using glob-def by simp
 hence ∀ x≥the-enat ⟨c #enat ninf-llist t⟩. γ (lnth (πcinf-llist t @l inf-llist t′)) x
  using validCE-act ‹∃ i≥n. ‖c‖t i› by blast
 ultimately have γ (lnth ((πc(inf-llist t)) @l (inf-llist t′))) (the-enat (⟨c #n′ inf-llist t⟩)) by simp
 with ‹∃ i≥n′. ‖c‖t i› show ?thesis using validCI-act by blast
next
 assume ¬(∃ i≥n′. ‖c‖t i)
 from ‹eval c t t′ n (□b(γ))› have eval c t t′ n (λt n. ∀ n′≥n. γ t n′) using glob-def by simp
 hence ∀ x≥the-enat ⟨c #enat ninf-llist t⟩. γ (lnth (πcinf-llist t @l inf-llist t′)) x
  using validCE-act ‹∃ i≥n. ‖c‖t i› by blast
 moreover have c↓t(n′) ≥ the-enat (⟨c #n inf-llist t⟩)
 proof −
  have ⟨c #n inf-llist t⟩≤llength (πc(inf-llist t)) using nAct-le-proj by metis
  moreover from ‹¬ (∃ i≥n′. ‖c‖t i)› have llength (πc(inf-llist t))≠∞
   by (metis llength-eq-infty-conv-lfinite lnth-inf-llist proj-finite2)
  ultimately have the-enat(⟨c #n inf-llist t⟩)≤the-enat(llength (πc(inf-llist t))) by simp
  moreover from ‹∃ i≥n. ‖c‖t i› ‹¬ (∃ i≥n′. ‖c‖t i)› have n′>⟨c ∧ t⟩
   using lActive-active by (meson leI le-eq-less-or-eq)
  hence c↓t(n′) > the-enat (llength (πc(inf-llist t))) − 1 using cnf2bhv-greater-llength by simp
  ultimately show ?thesis by simp
 qed
 ultimately have γ (lnth ((πc(inf-llist t)) @l (inf-llist t′))) (c↓t(n′)) by simp
 with ‹∃ i≥n. ‖c‖t i› ‹¬(∃ i≥n′. ‖c‖t i)› show ?thesis using validCI-cont by blast
qed
```

lemma *globEANow*:
fixes $c\ t\ t'\ n\ i\ \gamma$
assumes $n \leq i$
and $\|c\|_{t\ i}$
and *eval* $c\ t\ t'\ n\ (\Box_b\gamma)$
shows *eval* $c\ t\ t'\ i\ \gamma$
proof −
from ‹$\|c\|_{t\ i}$› ‹$n \leq i$› **have** $\exists\, i{\geq}n.\ \|c\|_{t\ i}$ **by** *auto*
moreover from ‹$n \leq i$› **have** $\langle c \Leftarrow t\rangle_n \leq i$ **using** *dual-order.trans lNactLe* **by** *blast*
ultimately show *?thesis* **using** *globEA*[*of* $n\ c\ t\ t'\ \gamma\ i$] ‹*eval* $c\ t\ t'\ n\ (\Box_b\gamma)$› **by** *simp*
qed

lemma *globEN*[*elim*]:
fixes c::*'id*
and t::*nat* ⇒ *cnf*
and t'::*nat* ⇒ *'cmp*
and n::*nat*
and n'::*nat*
assumes $\neg(\exists\, i{\geq}n.\ \|c\|_{t\ i})$
and *eval* $c\ t\ t'\ n\ (\Box_b(\gamma))$
and $n'{\geq}n$
shows *eval* $c\ t\ t'\ n'\ \gamma$
proof *cases*
assume $\exists\, i.\ \|c\|_{t\ i}$
moreover from ‹*eval* $c\ t\ t'\ n\ (\Box_b(\gamma))$› **have** *eval* $c\ t\ t'\ n\ (\lambda t\ n.\ \forall\, n'{\geq}n.\ \gamma\ t\ n')$
using *glob-def* **by** *simp*
ultimately have $\forall\, x{\geq}_{c\downarrow t}n.\ \gamma\ (lnth\ (\pi_c inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ x$
using *validCE-cont*[*of* $c\ t\ n\ t'\ \lambda t\ n.\ \forall\, n'{\geq}n.\ \gamma\ t\ n'$] ‹$\neg(\exists\, i{\geq}n.\ \|c\|_{t\ i})$› **by** *blast*
moreover from ‹$n'{\geq}n$› **have** ${}_{c\downarrow t}(n') \geq {}_{c\downarrow t}(n)$ **using** *cnf2bhv-mono* **by** *simp*
ultimately have $\gamma\ (lnth\ ((\pi_c(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ ({}_{c\downarrow t}(n'))$ **by** *simp*
moreover from ‹$\neg(\exists\, i{\geq}n.\ \|c\|_{t\ i})$› ‹$n'{\geq}n$› **have** $\neg(\exists\, i{\geq}n'.\ \|c\|_{t\ i})$ **by** *simp*
ultimately show *?thesis* **using** *validCI-cont* ‹$\exists\, i.\ \|c\|_{t\ i}$› **by** *blast*
next
assume $\neg(\exists\, i.\ \|c\|_{t\ i})$
moreover from ‹*eval* $c\ t\ t'\ n\ (\Box_b(\gamma))$› **have** *eval* $c\ t\ t'\ n\ (\lambda t\ n.\ \forall\, n'{\geq}n.\ \gamma\ t\ n')$
using *glob-def* **by** *simp*
ultimately have $\forall\, n'{\geq}n.\ \gamma\ (lnth\ (\pi_c inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ n'$
using ‹$\neg(\exists\, i.\ \|c\|_{t\ i})$› *validCE-not-act*[**where** $\gamma{=}\lambda t\ n.\ \forall\, n'{\geq}n.\ \gamma\ t\ n'$] **by** *blast*
with ‹$\neg(\exists\, i.\ \|c\|_{t\ i})$› ‹$n'{\geq}n$› **show** *?thesis* **using** *validCI-not-act* **by** *blast*
qed

2.4.12 Until Operator

definition *until* :: (*'cmp bta*) ⇒ (*'cmp bta*) ⇒ (*'cmp bta*) (**infixl** $\mathfrak{U}_b$ *21*)
where $\gamma'\ \mathfrak{U}_b\ \gamma \equiv \lambda\ t\ n.\ \exists\, n''{\geq}n.\ \gamma\ t\ n'' \wedge (\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma'\ t\ n')$

lemma *untilIA*[*intro*]:
fixes c::*'id*
and t::*nat* ⇒ *cnf*
and t'::*nat* ⇒ *'cmp*
and n::*nat*
and n'::*nat*
assumes $\exists\, i{\geq}n.\ \|c\|_{t\ i}$
and $n'{\geq}\langle c \Leftarrow t\rangle_n$
and $[\![\exists\, i{\geq}n'.\ \|c\|_{t\ i}]\!] \Longrightarrow \exists\, n''{\geq}\langle c \Leftarrow t\rangle_{n'}.\ n''{\leq}\ \langle c \rightarrow t\rangle_{n'} \wedge$ *eval* $c\ t\ t'\ n''\ \gamma \wedge$
$(\forall\, n'''{\geq}\langle c \rightarrow t\rangle_n.\ n'''{<}\ \langle c \Leftarrow t\rangle_{n''}$

$\longrightarrow (\exists\, n''''{\geq}\langle c \Leftarrow t\rangle_{n'''}.\ n''''{\leq}\ \langle c \to t\rangle_{n'''} \wedge\ eval\ c\ t\ t'\ n''''\ \gamma'))$
and $[\![\neg(\exists\, i{\geq}n'.\ \|c\|_{t\ i})]\!] \Longrightarrow\ eval\ c\ t\ t'\ n'\ \gamma \wedge$
$(\forall\, n''{\geq}\langle c \to t\rangle_{n}.\ n''{<}\ n'$
$\longrightarrow ((\exists\, i{\geq}n''.\ \|c\|_{t\ i}) \wedge (\exists\, n'''{\geq}\langle c \Leftarrow t\rangle_{n''}.\ n'''{\leq}\ \langle c \to t\rangle_{n''} \wedge\ eval\ c\ t\ t'\ n'''\ \gamma')) \vee$
$(\neg(\exists\, i{\geq}n''.\ \|c\|_{t\ i}) \wedge\ eval\ c\ t\ t'\ n''\ \gamma'))$
shows $eval\ c\ t\ t'\ n\ (\gamma'\ \mathfrak{U}_b\ \gamma)$
proof *cases*
assume $\exists\, i'{\geq}n'.\ \|c\|_{t\ i'}$
with *assms(3)* **obtain** n'' **where** $n''{\geq}\langle c \Leftarrow t\rangle_{n'}$ **and** $n''{\leq}\ \langle c \to t\rangle_{n'}$ **and** $eval\ c\ t\ t'\ n''\ \gamma$ **and**
a1: $\forall\, n'''{\geq}\langle c \to t\rangle_{n}.\ n'''{<}\ \langle c \Leftarrow t\rangle_{n''}$
$\longrightarrow (\exists\, n''''{\geq}\langle c \Leftarrow t\rangle_{n'''}.\ n''''{\leq}\ \langle c \to t\rangle_{n'''} \wedge\ eval\ c\ t\ t'\ n''''\ \gamma')$ **by** *blast*
hence $\exists\, i'{\geq}n''.\ \|c\|_{t\ i'}$ **using** $\langle\exists\, i'{\geq}n'.\ \|c\|_{t\ i'}\rangle$ *nxtActI* **by** *blast*
with $\langle eval\ c\ t\ t'\ n''\ \gamma\rangle$ **have**
$\gamma\ (lnth\ ((\pi_c(inf\text{-}llist\ t))\ @_l\ (inf\text{-}llist\ t')))\ (the\text{-}enat\ (\langle c\ \#_{n''}\ inf\text{-}llist\ t\rangle))$
using *validCE-act* **by** *blast*
moreover have $the\text{-}enat\ (\langle c\ \#_n\ inf\text{-}llist\ t\rangle) \leq\ the\text{-}enat\ (\langle c\ \#_{n''}\ inf\text{-}llist\ t\rangle)$
proof −
from $\langle\langle c \Leftarrow t\rangle_n{\leq}n'\rangle$ **have** $\langle c\ \#_n\ inf\text{-}llist\ t\rangle \leq \langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle$
using *nAct-mono-lNact* **by** *simp*
moreover from $\langle\langle c \Leftarrow t\rangle_{n'}{\leq}n''\rangle$ **have** $\langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle \leq \langle c\ \#_{n''}\ inf\text{-}llist\ t\rangle$
using *nAct-mono-lNact* **by** *simp*
ultimately have $\langle c\ \#_n\ inf\text{-}llist\ t\rangle \leq \langle c\ \#_{n''}\ inf\text{-}llist\ t\rangle$ **by** *simp*
moreover have $\langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle \neq \infty$ **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
moreover have $\exists\, i'{\geq}n.\ \|c\|_{t\ i'}$
proof −
from $\langle\exists\, i'{\geq}n'.\ \|c\|_{t\ i'}\rangle$ **obtain** i' **where** $i'{\geq}n'$ **and** $\|c\|_{t\ i'}$ **by** *blast*
with $\langle n'{\geq}\langle c \Leftarrow t\rangle_n\rangle$ **have** $i'{\geq}\ n$ **using** *lNactGe le-trans* **by** *blast*
with $\langle\|c\|_{t\ i'}\rangle$ **show** *?thesis* **by** *blast*
qed
moreover have $\forall\, n'{\geq}the\text{-}enat\ \langle c\ \#_n inf\text{-}llist\ t\rangle.\ n' < (the\text{-}enat\ \langle c\ \#_{enat\ n''} inf\text{-}llist\ t\rangle)$
$\longrightarrow \gamma'\ (lnth\ (\pi_c inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ n'$
proof
fix $x{::}nat$ **show** $x{\geq}the\text{-}enat\ (\langle c\ \#_n\ inf\text{-}llist\ t\rangle)$
$\longrightarrow x < (the\text{-}enat\ \langle c\ \#_{enat\ n''} inf\text{-}llist\ t\rangle) \longrightarrow \gamma'\ (lnth\ (\pi_c inf\text{-}llist\ t\ @_l\ inf\text{-}llist\ t'))\ x$
proof (*rule HOL.impI*[*OF HOL.impI*])
assume $x{\geq}the\text{-}enat\ (\langle c\ \#_n\ inf\text{-}llist\ t\rangle)$ **and** $x < (the\text{-}enat\ \langle c\ \#_{enat\ n''} inf\text{-}llist\ t\rangle)$
moreover have $the\text{-}enat\ (\langle c\ \#_{enat\ n''}\ inf\text{-}llist\ t\rangle) = \langle c\ \#_{enat\ n''}\ inf\text{-}llist\ t\rangle$ **by** *simp*
ultimately have $x{<}llength\ (\pi_c(inf\text{-}llist\ t))$ **using** *nAct-le-proj*[*of c n'' inf-llist t*]
by (*metis enat-ord-simps(2) less-le-trans*)
hence $x{<}llength\ (\pi_c(inf\text{-}llist\ t))$ **by** *simp*
then obtain $n'{::}nat$ **where** $x{=}\langle c\ \#_{n'}\ inf\text{-}llist\ t\rangle$ **using** *nAct-exists* **by** *blast*
moreover from $\langle enat\ x < llength\ (\pi_c(inf\text{-}llist\ t))\rangle\ \langle enat\ x = \langle c\ \#_{enat\ n'}\ inf\text{-}llist\ t\rangle\rangle$
have $\exists\, i{\geq}n'.\ \|c\|_{t\ i}$ **using** *nAct-less-llength-active* **by** *force*
then obtain i **where** $i{\geq}n'$ **and** $\|c\|_{t\ i}$ **and** $\neg\ (\exists\, k{\geq}n'.\ k < i \wedge \|c\|_{t\ k})$ **using** *nact-exists* **by** *blast*
moreover have $enat\ i - 1 < llength\ (inf\text{-}llist\ t)$ **by** (*simp add*: *one-enat-def*)
ultimately have $x{=}\langle c\ \#_i\ inf\text{-}llist\ t\rangle$ **using** *one-enat-def nAct-not-active-same* **by** *simp*
moreover have $\langle c\ \#_i\ inf\text{-}llist\ t\rangle{\neq}\infty$ **by** *simp*
ultimately have $x{=}the\text{-}enat(\langle c\ \#_i\ inf\text{-}llist\ t\rangle)$ **by** *fastforce*
from $\langle x{\geq}the\text{-}enat\ (\langle c\ \#_n\ inf\text{-}llist\ t\rangle)\rangle\ \langle x{=}the\text{-}enat(\langle c\ \#_i\ inf\text{-}llist\ t\rangle)\rangle$
have $the\text{-}enat\ (\langle c\ \#_i\ inf\text{-}llist\ t\rangle){\geq}the\text{-}enat\ (\langle c\ \#_n\ inf\text{-}llist\ t\rangle)$ **by** *simp*
with $\langle\|c\|_{t\ i}\rangle$ **have** $i{\geq}\langle c \to t\rangle_n$ **using** *active-geq-nxtAct* **by** *simp*
moreover have $i < \langle c \Leftarrow t\rangle_{n''}$
proof −

have *the-enat* $\langle c\ \#_{enat\ n''} inf\text{-}llist\ t\rangle = \langle c\ \#_{enat\ n''} inf\text{-}llist\ t\rangle$ **by** *simp*
with $\langle x < (the\text{-}enat\ \langle c\ \#_{enat\ n''} inf\text{-}llist\ t\rangle)\rangle$ **and** $\langle x{=}\langle c\ \#_i\ inf\text{-}llist\ t\rangle\rangle$ **have**
$\langle c\ \#_i\ inf\text{-}llist\ t\rangle{<}\langle c\ \#_{n''}\ inf\text{-}llist\ t\rangle$ **by** (*metis enat-ord-simps*(2))
hence $i{<}n''$ **using** *nAct-strict-mono-back*[*of c i inf-llist t n''*] **by** *auto*
with $\langle \|c\|_{t\ i}\rangle$ **show** *?thesis* **using** *lNact-notActive leI* **by** *blast*
qed
ultimately obtain n'' **where** *eval c t t' n'' γ'* **and** $n''{\geq}\langle c \Leftarrow t\rangle_i$ **and** $n''{\leq}\langle c \rightarrow t\rangle_i$
using *a1* **by** *auto*
moreover have $\exists\, i'{\geq}n''.\ \|c\|_{t\ i'}$
using $\langle \|c\|_{t\ i}\rangle$ $\langle n''{\leq}\langle c \rightarrow t\rangle_i\rangle$ *less-or-eq-imp-le nxtAct-active* **by** *auto*
ultimately have γ' (*lnth* ((π*c*(*inf-llist t*)) @*l* (*inf-llist t'*))) (*the-enat* ($\langle c\ \#_{n''}$ *inf-llist t*⟩))
using *validCE-act*[*of n'' c t t' γ'*] **by** *blast*
moreover from $\langle n''{\geq}\langle c \Leftarrow t\rangle_i\rangle$ **and** $\langle n''{\leq}\langle c \rightarrow t\rangle_i\rangle$
have *the-enat* ($\langle c\ \#_{n''}$ *inf-llist t*⟩)=*the-enat* ($\langle c\ \#_i$ *inf-llist t*⟩) **using** *nAct-same* **by** *simp*
hence *the-enat* ($\langle c\ \#_{n''}$ *inf-llist t*⟩) = *x* **by** (*simp add*: $\langle x =$ *the-enat* $\langle c\ \#_{enat\ i} inf\text{-}llist\ t\rangle\rangle$)
ultimately show γ' (*lnth* ((π*c*(*inf-llist t*)) @*l* (*inf-llist t'*))) *x* **by** *simp*
qed
qed
ultimately have *eval c t t' n* (λ *t n*. ∃ *n''*≥*n*. *γ t n''* ∧ (∀ *n'*≥*n*. *n'* < *n''* ⟶ *γ' t n'*))
using *validCI-act*[**where** *γ*=λ *t n*. ∃ *n''*≥*n*. *γ t n''* ∧ (∀ *n'*≥*n*. *n'* < *n''* ⟶ *γ' t n'*)] **by** *blast*
thus *?thesis* **using** *until-def* **by** *simp*
next
assume ¬(∃ *i'*≥*n'*. $\|c\|_{t\ i'}$)
with *assms*(4) **have** *eval c t t' n' γ* **and** *a2*: ∀ *n''*≥⟨*c* → *t*⟩*n*. *n''*< *n'*
⟶ ((∃ *i*≥*n''*. $\|c\|_{t\ i}$) ∧ (∃ *n'''*≥$\langle c \Leftarrow t\rangle_{n''}$. *n'''*≤ $\langle c \rightarrow t\rangle_{n''}$ ∧ *eval c t t' n''' γ'*)) ∨
(¬(∃ *i*≥*n''*. $\|c\|_{t\ i}$) ∧ *eval c t t' n'' γ'*) **by** *auto*
with $\langle \neg(\exists\, i'{\geq}n'.\ \|c\|_{t\ i'})\rangle$ $\langle$*eval c t t' n' γ*⟩ $\langle \exists\, i{\geq}n.\ \|c\|_{t\ i}\rangle$ **have**
γ (*lnth* ((π*c*(*inf-llist t*)) @*l* (*inf-llist t'*))) (*c*↓*t*(*n'*)) **using** *validCE-cont* **by** *blast*
moreover have *c*↓*t*(*n'*) ≥ *the-enat* ($\langle c\ \#_n$ *inf-llist t*⟩)
proof −
from $\langle(\exists\, i{\geq}n.\ \|c\|_{t\ i})\rangle$ $\langle \neg(\exists\, i'{\geq}n'.\ \|c\|_{t\ i'})\rangle$ **have** $n' \geq \langle c \wedge t\rangle$ **using** *lActive-less* **by** *auto*
hence *c*↓*t*(*n'*) ≥ *the-enat* (*llength* (π*c*(*inf-llist t*))) − *1* **using** *cnf2bhv-ge-llength* **by** *simp*
moreover have *the-enat*(*llength* (π*c*(*inf-llist t*))) − *1* ≥ *the-enat*($\langle c\ \#_n$ *inf-llist t*⟩)
proof −
from $\langle \exists\, i{\geq}n.\ \|c\|_{t\ i}\rangle$ **have** *llength* (π*c*(*inf-llist t*)) ≥ *eSuc* ($\langle c\ \#_n$ *inf-llist t*⟩)
using *nAct-llength-proj* **by** *simp*
moreover from $\langle \neg(\exists\, i'{\geq}n'.\ \|c\|_{t\ i'})\rangle$ **have** *lfinite* (π*c*(*inf-llist t*))
using *proj-finite2*[*of inf-llist t*] **by** *simp*
hence *llength* (π*c*(*inf-llist t*))≠∞ **using** *llength-eq-infty-conv-lfinite* **by** *auto*
ultimately have *the-enat* (*llength* (π*c*(*inf-llist t*))) ≥ *the-enat*(*eSuc* ($\langle c\ \#_n$ *inf-llist t*⟩))
by *simp*
moreover have $\langle c\ \#_n$ *inf-llist t*⟩≠∞ **by** *simp*
ultimately have *the-enat* (*llength* (π*c*(*inf-llist t*))) ≥ *Suc* (*the-enat* ($\langle c\ \#_n$ *inf-llist t*⟩))
using *the-enat-eSuc* **by** *simp*
thus *?thesis* **by** *simp*
qed
ultimately show *?thesis* **by** *simp*
qed
moreover have ∀ *x*≥*the-enat* $\langle c\ \#_n inf\text{-}llist\ t\rangle$. *x* < (*c*↓*t*(*n'*))
⟶ *γ'* (*lnth* (π*c inf-llist t* @*l inf-llist t'*)) *x*
proof
fix *x*::*nat* **show**
x≥*the-enat* $\langle c\ \#_n inf\text{-}llist\ t\rangle$ ⟶ *x* < (*c*↓*t*(*n'*)) ⟶ *γ'* (*lnth* (π*c inf-llist t* @*l inf-llist t'*)) *x*
proof (*rule HOL.impI*[*OF HOL.impI*])
assume *x*≥*the-enat* $\langle c\ \#_n inf\text{-}llist\ t\rangle$ **and** *x* < (*c*↓*t*(*n'*))

show γ' (*lnth* ((*π*$_c$(*inf-llist t*)) @$_l$ (*inf-llist t'*))) *x*
proof (*cases*)
assume (*x* ≥ *llength* (*π*$_c$(*inf-llist t*)))
hence *lfinite* (*π*$_c$(*inf-llist t*))
using *llength-geq-enat-lfiniteD*[*of* *π*$_c$(*inf-llist t*) *x*] **by** *simp*
then obtain *z* **where** ∀ *n''*>*z*. ¬ ‖*c*‖$_{t\ n''}$ **using** *proj-finite-bound* **by** *blast*
moreover have ‖*c*‖$_{t\ \langle c \to t\rangle_n}$ **by** (*simp add*: ‹∃ *i*≥*n*. ‖*c*‖$_{t\ i}$› *nxtActI*)
ultimately have ⟨*c* ∧ *t*⟩≥⟨*c* → *t*⟩$_n$ **using** *lActive-greatest*[*of* *c* *t* ⟨*c* → *t*⟩$_n$] **by** *blast*
moreover have $_c\uparrow_t(x)$ ≥ ⟨*c* ∧ *t*⟩ **by** *simp*
ultimately have $_c\uparrow_t(x)$ ≥ ⟨*c* → *t*⟩$_n$ **by** *arith*
moreover have ¬ (∃ *i'*≥$_c\uparrow_t(x)$. ‖*c*‖$_{t\ i'}$)
proof −
from ‹*lfinite* (*π*$_c$(*inf-llist t*))› ‹∃ *i*≥*n*. ‖*c*‖$_{t\ i}$›
have $_c\uparrow_t$(*the-enat* (*llength* (*π*$_c$(*inf-llist t*)))) = *Suc* (⟨*c* ∧ *t*⟩)
using *bhv2cnf-lActive* **by** *blast*
moreover from ‹(*x* ≥ *llength* (*π*$_c$(*inf-llist t*)))› **have** *x* ≥ *the-enat*(*llength* (*π*$_c$(*inf-llist t*)))
using *the-enat-mono* **by** *fastforce*
hence $_c\uparrow_t(x)$ ≥ $_c\uparrow_t$(*the-enat* (*llength* (*π*$_c$(*inf-llist t*))))
using *bhv2cnf-mono*[*of the-enat* (*llength* (*π*$_c$(*inf-llist t*))) *x*] **by** *simp*
ultimately have $_c\uparrow_t(x)$ ≥ *Suc* (⟨*c* ∧ *t*⟩) **by** *simp*
hence $_c\uparrow_t(x)$ > ⟨*c* ∧ *t*⟩ **by** *simp*
with ‹∀ *n''*>*z*. ¬ ‖*c*‖$_{t\ n''}$› **show** *?thesis* **using** *lActive-greater-active-all* **by** *simp*
qed
moreover have $_c\uparrow_t x$ < *n'*
proof −
from ‹*lfinite* (*π*$_c$(*inf-llist t*))› **have** *llength* (*π*$_c$*inf-llist t*) = *the-enat* (*llength* (*π*$_c$*inf-llist t*))
by (*simp add*: *enat-the-enat llength-eq-infty-conv-lfinite*)
with ‹*x* ≥ *llength* (*π*$_c$(*inf-llist t*))› **have** *x*≥*the-enat* (*llength* (*π*$_c$*inf-llist t*))
using *enat-ord-simps*(*1*) **by** *fastforce*
moreover from ‹∃ *i*≥*n*. ‖*c*‖$_{t\ i}$› **have** *llength* (*π*$_c$*inf-llist t*)≥*1* **using** *proj-one* **by** *force*
ultimately have *the-enat* (*llength* (*π*$_c$*inf-llist t*)) − *1* ≤ *x* **by** *simp*
with ‹*x* < ($_c\downarrow_t$(*n'*))› **show** *?thesis* **using** *c2p-mono-p2c-strict* **by** *simp*
qed
ultimately have *eval* *c* *t* *t'* ($_c\uparrow_t$(*x*)) *γ'* **using** *a2* **by** *blast*
hence *γ'* (*lnth* ((*π*$_c$(*inf-llist t*)) @$_l$ (*inf-llist t'*))) ($_c\downarrow_t(_c\uparrow_t(x))$)
using *validCE-cont*[*of* *c* *t* $_c\uparrow_t(x)$ *t'* *γ'*] ‹∃ *i*≥*n*. ‖*c*‖$_{t\ i}$› ‹¬ (∃ *i'*≥$_c\uparrow_t(x)$. ‖*c*‖$_{t\ i'}$)› **by** *blast*
moreover from ‹(*x* ≥ *llength* (*π*$_c$(*inf-llist t*)))›
have (*enat x* ≥ *llength* (*π*$_c$(*inf-llist t*))) **by** *auto*
with ‹*lfinite* (*π*$_c$(*inf-llist t*))› **have** *llength* (*π*$_c$(*inf-llist t*))≠∞
using *llength-eq-infty-conv-lfinite* **by** *auto*
with ‹(*x* ≥ *llength* (*π*$_c$(*inf-llist t*)))›
have *the-enat*(*llength* (*π*$_c$(*inf-llist t*))) − *1* ≤ *x* **by** *auto*
ultimately show *?thesis* **using** *cnf2bhv-bhv2cnf*[*of* *c* *t* *x*] **by** *simp*
next
assume ¬(*x* ≥ *llength* (*π*$_c$(*inf-llist t*)))
hence *x*<*llength* (*π*$_c$(*inf-llist t*)) **by** *simp*
then obtain *n''*::*nat* **where** *x*=⟨*c* #$_{n''}$ *inf-llist t*⟩ **using** *nAct-exists* **by** *blast*
moreover from ‹*enat x* < *llength* (*π*$_c$(*inf-llist t*))› ‹*enat x* = ⟨*c* #$_{enat\ n''}$ *inf-llist t*⟩›
have ∃ *i*≥*n''*. ‖*c*‖$_{t\ i}$ **using** *nAct-less-llength-active* **by** *force*
then obtain *i* **where** *i*≥*n''* **and** ‖*c*‖$_{t\ i}$ **and** ¬ (∃ *k*≥*n''*. *k* < *i* ∧ ‖*c*‖$_{t\ k}$)
using *nact-exists* **by** *blast*
moreover have *enat i* − *1* < *llength* (*inf-llist t*) **by** (*simp add*: *one-enat-def*)
ultimately have *x*=⟨*c* #$_i$ *inf-llist t*⟩ **using** *one-enat-def nAct-not-active-same* **by** *simp*
moreover have ⟨*c* #$_i$ *inf-llist t*⟩≠∞ **by** *simp*
ultimately have *x*=*the-enat*(⟨*c* #$_i$ *inf-llist t*⟩) **by** *fastforce*

```
      from ‹x≥the-enat (⟨c #n inf-llist t⟩)› ‹x=the-enat(⟨c #i inf-llist t⟩)›
      have the-enat (⟨c #i inf-llist t⟩)≥the-enat (⟨c #n inf-llist t⟩) by simp
      with ‹‖c‖t i› have i≥⟨c → t⟩n using active-geq-nxtAct by simp
      moreover from ‹x=⟨c #i inf-llist t⟩› ‹x < llength (πc(inf-llist t))›
        have ∃ i'. i ≤ enat i' ∧ ‖c‖t i' using nAct-less-llength-active[of x c inf-llist t i] by simp
      hence ∃ i'≥i. ‖c‖t i' by simp
      moreover have i<n'
      proof −
        from ‹∃ i≥n. ‖c‖t i› ‹¬(∃ i'≥n'. ‖c‖t i')› have n'≥⟨c ∧ t⟩ using lActive-less by auto
        hence c↓t(n')≥the-enat(llength (πc(inf-llist t))) − 1 using cnf2bhv-ge-llength by simp
        with ‹x<llength (πc(inf-llist t))› show ?thesis
          using ‹¬ (∃ i'≥n'. ‖c‖t i')› ‹‖c‖t i› le-neq-implies-less nat-le-linear by blast
      qed
      ultimately obtain n''' where eval c t t' n''' γ' and n'''≥⟨c ⇐ t⟩i and n'''≤⟨c → t⟩i
        using a2 by blast
      moreover from ‹‖c‖t i› have ‖c‖t ⟨c → t⟩i using nxtActI by auto
      with ‹n'''≤⟨c → t⟩i› have ∃ i'≥n'''. ‖c‖t i' using less-or-eq-imp-le by blast
      ultimately have γ' (lnth ((πc(inf-llist t)) @l (inf-llist t'))) (the-enat (⟨c #n''' inf-llist t⟩))
        using validCE-act[of n''' c t t' γ'] by blast
      moreover from ‹n'''≥⟨c ⇐ t⟩i› and ‹n'''≤⟨c → t⟩i›
        have the-enat (⟨c #n''' inf-llist t⟩)=the-enat (⟨c #i inf-llist t⟩) using nAct-same by simp
      hence the-enat (⟨c #n''' inf-llist t⟩) = x by (simp add: ‹x = the-enat ⟨c #enat i inf-llist t⟩›)
      ultimately have γ' (lnth ((πc(inf-llist t)) @l (inf-llist t'))) (the-enat x) by simp
      thus ?thesis by simp
    qed
  qed
 qed
 ultimately have eval c t t' n (λ t n. ∃ n''≥n. γ t n'' ∧ (∀ n'≥n. n' < n'' ⟶ γ' t n'))
  using ‹∃ i≥n. ‖c‖t i› validCI-act[of n c t λ t n. ∃ n''≥n. γ t n'' ∧ (∀ n'≥n. n' < n'' ⟶ γ' t n') t']
  by blast
 thus ?thesis using until-def by simp
qed

lemma untilIN[intro]:
 fixes c::'id
  and t::nat ⇒ cnf
  and t'::nat ⇒ 'cmp
  and n::nat
  and n'::nat
 assumes ¬(∃ i≥n. ‖c‖t i)
  and n'≥n
  and eval c t t' n' γ
  and a1: ⋀n''. ⟦n≤n''; n''<n'⟧ ⟹ eval c t t' n'' γ'
 shows eval c t t' n (γ' 𝔘b γ)
proof cases
 assume ∃ i. ‖c‖t i
 moreover from assms(1) assms(2) have ¬(∃ i'≥n'. ‖c‖t i') by simp
 ultimately have γ (lnth ((πc(inf-llist t)) @l (inf-llist t'))) (c↓t(n'))
  using validCE-cont[of c t n' t' γ] ‹eval c t t' n' γ› by blast
 moreover from ‹n'≥n› have c↓t(n') ≥ c↓t(n) using cnf2bhv-mono by simp
 moreover have ∀ x::nat≥ c↓t(n). x<c↓t(n') ⟶ γ' (lnth ((πc(inf-llist t)) @l (inf-llist t'))) x
 proof (rule HOL.allI[OF HOL.impI[OF HOL.impI]])
  fix x assume x≥c↓t(n) and x<c↓t(n')

  from ‹¬(∃ i≥n. ‖c‖t i)› have ⟨c ∧ t⟩ ≤ n using ‹∃ i. ‖c‖t i› lActive-less by auto
```

with ‹$x \geq_c \downarrow_t(n)$› **have** $_c\uparrow_t(x) \geq n$ **using** *p2c-mono-c2p* **by** *simp*
moreover from ‹$\langle c \wedge t\rangle \leq n$› ‹$_c\downarrow_t(n) \leq x$› **have** $x \geq$ *the-enat* (*llength* (π_c(*inf-llist t*))) − *1*
using *cnf2bhv-ge-llength dual-order.trans* **by** *blast*
with ‹$x <_c \downarrow_t(n')$› **have** $_c\uparrow_t(x) < n'$ **using** *c2p-mono-p2c-strict*[*of c t x n'*] **by** *simp*
moreover from ‹$\neg$ ($\exists\, i \geq n.$ $\|c\|_{t\ i}$)› ‹$_c\uparrow_t(x) \geq n$› **have** $\neg$ ($\exists\, i'' \geq_c \uparrow_t(x).$ $\|c\|_{t\ i''}$) **by** *auto*
ultimately have *eval c t t'* ($_c\uparrow_t(x)$) γ' **using** *a1*[*of* $_c\uparrow_t(x)$] **by** *simp*
with ‹$\neg$ ($\exists\, i'' \geq_c \uparrow_t x.$ $\|c\|_{t\ i''}$)›
have γ' (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*))) ($_c\downarrow_t(_c\uparrow_t(x))$)
using *validCE-cont*[*of c t* $_c\uparrow_t(x)$ *t' γ'*] ‹$\exists\, i.$ $\|c\|_{t\ i}$› **by** *blast*
moreover have $x \geq$ *the-enat* (*llength* (π_c(*inf-llist t*))) − *1*
using ‹$_c\downarrow_t(n) \leq x$› *cnf2bhv-def* **by** *auto*
ultimately show γ' (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*))) (*x*)
using *cnf2bhv-bhv2cnf* **by** *simp*
qed
ultimately have *eval c t t' n* (λ *t n.* $\exists\, n'' \geq n.$ γ *t n''* $\wedge$ ($\forall\, n' \geq n.$ $n' < n'' \longrightarrow \gamma'$ *t n'*))
using *validCI-cont*[*of c t n* λ *t n.* $\exists\, n'' \geq n.$ γ *t n''* $\wedge$ ($\forall\, n' \geq n.$ $n' < n'' \longrightarrow \gamma'$ *t n'*) *t'*]
‹$\exists\, i.$ $\|c\|_{t\ i}$› ‹$\neg(\exists\, i' \geq n.$ $\|c\|_{t\ i'}$)› **by** *blast*
thus *?thesis* **using** *until-def* **by** *simp*
next
assume $\neg(\exists\, i.$ $\|c\|_{t\ i})$
with *assms* **have** $\exists\, n'' \geq n.$ γ (*lnth* (π_c*inf-llist t* @$_l$ *inf-llist t'*)) *n''* $\wedge$
($\forall\, n' \geq n.$ $n' < n'' \longrightarrow \gamma'$ (*lnth* (π_c*inf-llist t* @$_l$ *inf-llist t'*)) *n'*) **using** *validCE-not-act* **by** *blast*
with ‹$\neg(\exists\, i.$ $\|c\|_{t\ i}$)› **have** *eval c t t' n* (λ *t n.* $\exists\, n'' \geq n.$ γ *t n''* $\wedge$ ($\forall\, n' \geq n.$ $n' < n'' \longrightarrow \gamma'$ *t n'*))
using *validCI-not-act*[**where** γ=λ *t n.* $\exists\, n'' \geq n.$ γ *t n''* $\wedge$ ($\forall\, n' \geq n.$ $n' < n'' \longrightarrow \gamma'$ *t n'*)] **by** *blast*
thus *?thesis* **using** *until-def* **by** *simp*
qed

lemma *untilEA*[*elim*]:
fixes *n*::*nat*
and *n'*::*nat*
and *t*::*nat* ⇒ *cnf*
and *t'*::*nat* ⇒ *'cmp*
and *c*::*'id*
assumes $\exists\, i \geq n.$ $\|c\|_{t\ i}$
and *eval c t t' n* (γ' $\mathfrak{U}_b$ γ)
shows $\exists\, n' \geq \langle c \to t\rangle_n.$
(($\exists\, i \geq n'.$ $\|c\|_{t\ i}$) $\wedge$ ($\forall\, n'' \geq \langle c \Leftarrow t\rangle_{n'}.$ $n'' \leq \langle c \to t\rangle_{n'} \longrightarrow$ *eval c t t' n''* γ)
$\wedge$ ($\forall\, n'' \geq \langle c \Leftarrow t\rangle_n.$ $n'' < \langle c \Leftarrow t\rangle_{n'} \longrightarrow$ *eval c t t' n''* γ') $\vee$
($\neg(\exists\, i \geq n'.$ $\|c\|_{t\ i}$)) $\wedge$ *eval c t t' n'* γ $\wedge$ ($\forall\, n'' \geq \langle c \Leftarrow t\rangle_n.$ $n'' < n' \longrightarrow$ *eval c t t' n''* γ'))
proof −
from ‹*eval c t t' n* (γ' $\mathfrak{U}_b$ γ)›
have *eval c t t' n* (λ *t n.* $\exists\, n'' \geq n.$ γ *t n''* $\wedge$ ($\forall\, n' \geq n.$ $n' < n'' \longrightarrow \gamma'$ *t n'*)) **using** *until-def* **by** *simp*
with ‹$\exists\, i \geq n.$ $\|c\|_{t\ i}$› **obtain** *x*
where *x*≥*the-enat* ⟨*c* #$_{enat}$ *n inf-llist t*⟩ **and** γ (*lnth* (π_c*inf-llist t* @$_l$ *inf-llist t'*)) *x*
and *a1*: $\forall\, x' \geq$*the-enat* ⟨*c* #$_{enat}$ *n inf-llist t*⟩. $x' < x \longrightarrow \gamma'$ (*lnth* (π_c*inf-llist t* @$_l$ *inf-llist t'*)) x'
using *validCE-act*[**where** γ=λ *t n.* $\exists\, n'' \geq n.$ γ *t n''* $\wedge$ ($\forall\, n' \geq n.$ $n' < n'' \longrightarrow \gamma'$ *t n'*)] **by** *blast*
thus *?thesis*
proof (*cases*)
assume *x* ≥ *llength* (π_c(*inf-llist t*))
moreover from ‹(*x* ≥ *llength* (π_c(*inf-llist t*)))› **have** *llength* (π_c(*inf-llist t*))≠∞
by (*metis infinity-ileE*)
moreover from ‹$\exists\, i \geq n.$ $\|c\|_{t\ i}$› **have** *llength* (π_c(*inf-llist t*))≥*1*
using *proj-one*[*of inf-llist t*] **by** *auto*
ultimately have *the-enat* (*llength* (π_c(*inf-llist t*))) − *1* < *x*
by (*metis One-nat-def Suc-ile-eq antisym-conv2 diff-Suc-less enat-ord-simps*(*2*)

enat-the-enat less-imp-diff-less one-enat-def)
hence $x = {}_{c}{\downarrow_t}({}_{c}{\uparrow_t}(x))$ **using** *cnf2bhv-bhv2cnf* **by** *simp*
with ‹γ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*))) x›
have γ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*))) ($c{\downarrow_t}({}_{c}{\uparrow_t}(x))$) **by** *simp*
moreover have $\neg(\exists\, i{\geq}{}_{c}{\uparrow_t}(x).\ \|c\|_{t\ i})$
proof −
from ‹$x \geq$ *llength* (π_c(*inf-llist t*))› **have** *lfinite* (π_c(*inf-llist t*))
using *llength-geq-enat-lfiniteD*[*of* π_c(*inf-llist t*) x] **by** *simp*
then obtain z **where** $\forall\, n''{>}z.\ \neg\ \|c\|_{t\ n''}$ **using** *proj-finite-bound* **by** *blast*
moreover from ‹*the-enat* (*llength* (π_c(*inf-llist t*))) − *1* < x› **have** $\langle c \wedge t\rangle < {}_{c}{\uparrow_t}(x)$
using *bhv2cnf-greater-lActive* **by** *simp*
ultimately show *?thesis* **using** *lActive-greater-active-all* **by** *simp*
qed
ultimately have *eval c t t'* (${}_{c}{\uparrow_t}x$) γ
using ‹$\exists\, i{\geq}n.\ \|c\|_{t\ i}$› *validCI-cont*[*of c t* ${}_{c}{\uparrow_t}(x)$] **by** *blast*
moreover have ${}_{c}{\uparrow_t}(x) \geq \langle c \rightarrow t\rangle_n$
proof −
from ‹$x \geq$ *llength* (π_c(*inf-llist t*))› **have** *lfinite* (π_c(*inf-llist t*))
using *llength-geq-enat-lfiniteD*[*of* π_c(*inf-llist t*) x] **by** *simp*
then obtain z **where** $\forall\, n''{>}z.\ \neg\ \|c\|_{t\ n''}$ **using** *proj-finite-bound* **by** *blast*
moreover from ‹$\exists\, i{\geq}n.\ \|c\|_{t\ i}$› **have** $\|c\|_{t\ \langle c \rightarrow t\rangle_n}$ **using** *nxtActI* **by** *simp*
ultimately have $\langle c \wedge t\rangle{\geq}\langle c \rightarrow t\rangle_n$ **using** *lActive-greatest* **by** *fastforce*
moreover have ${}_{c}{\uparrow_t}(x) \geq \langle c \wedge t\rangle$ **by** *simp*
ultimately show ${}_{c}{\uparrow_t}(x) \geq \langle c \rightarrow t\rangle_n$ **by** *arith*
qed
moreover have $\forall\, n''{\geq}\langle c \Leftarrow t\rangle_n.\ n'' < ({}_{c}{\uparrow_t}x) \longrightarrow$ *eval c t t' n'* γ'
proof
fix n'' **show** $\langle c \Leftarrow t\rangle_n \leq n'' \longrightarrow n'' < {}_{c}{\uparrow_t}x \longrightarrow$ *eval c t t' n'* γ'
proof (*rule HOL.impI*[*OF HOL.impI*])
assume $\langle c \Leftarrow t\rangle_n \leq n''$ **and** $n'' < {}_{c}{\uparrow_t}x$
show *eval c t t' n'* γ'
proof *cases*
assume $\exists\, i{\geq}n''.\ \|c\|_{t\ i}$
with ‹$n''{\geq}\langle c \Leftarrow t\rangle_n$› **have** *the-enat* ($\langle c\ \#_{n''}$ *inf-llist t*⟩) ≥ *the-enat* ($\langle c\ \#_n$ *inf-llist t*⟩)
using *nAct-mono-lNact* ‹$\exists\, i{\geq}n.\ \|c\|_{t\ i}$› **by** *simp*
moreover have *the-enat* ($\langle c\ \#_{n''}$ *inf-llist t*⟩)<x
proof −
from ‹$\exists\, i{\geq}n''.\ \|c\|_{t\ i}$› **have** *eSuc* $\langle c\ \#_{enat\ n''}$*inf-llist t*⟩ ≤ *llength* (π_c*inf-llist t*)
using *nAct-llength-proj* **by** *auto*
with ‹$x \geq$ *llength* (π_c(*inf-llist t*))› **have** *eSuc* $\langle c\ \#_{enat\ n''}$*inf-llist t*⟩ ≤ x **by** *simp*
moreover have $\langle c\ \#_{enat\ n''}$*inf-llist t*⟩$\neq\infty$ **by** *simp*
ultimately have *Suc* (*the-enat*($\langle c\ \#_{enat\ n''}$*inf-llist t*⟩)) ≤ x
by (*metis enat.distinct*(*2*) *the-enat.simps the-enat-eSuc the-enat-mono*)
thus *?thesis* **by** *simp*
qed
ultimately have γ' (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*))) (*the-enat* ($\langle c\ \#_{n''}$ *inf-llist t*⟩))
using *a1* **by** *auto*
with ‹$\exists\, i{\geq}n''.\ \|c\|_{t\ i}$› **show** *?thesis* **using** *validCI-act* **by** *blast*
next
assume $\neg(\exists\, i{\geq}n''.\ \|c\|_{t\ i})$
moreover have ${}_{c}{\downarrow_t}(n'')$ ≥ *the-enat* ($\langle c\ \#_n$ *inf-llist t*⟩)
proof −
have $\langle c\ \#_n$ *inf-llist t*⟩≤*llength* (π_c(*inf-llist t*)) **using** *nAct-le-proj* **by** *metis*
moreover from ‹$\neg$ ($\exists\, i{\geq}n''.\ \|c\|_{t\ i}$)› **have** *llength* ($\pi_c$(*inf-llist t*))$\neq\infty$
by (*metis llength-eq-infty-conv-lfinite lnth-inf-llist proj-finite2*)

ultimately have *the-enat*($\langle c \#_n$ *inf-llist* $t\rangle$)$\leq$*the-enat*(*llength* (π_c(*inf-llist* t))) **by** *simp*
moreover from ‹$\exists\, i{\geq}n.\ \|c\|_{t\ i}$› ‹$\neg\ (\exists\, i{\geq}n''.\ \|c\|_{t\ i})$› **have** $n''{>}\langle c \wedge t\rangle$
using *lActive-active* **by** (*meson leI le-eq-less-or-eq*)
hence $_c{\downarrow}_t(n'')$ > *the-enat* (*llength* (π_c(*inf-llist* t))) − *1* **using** *cnf2bhv-greater-llength* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
moreover from ‹$\neg(\exists\, i{\geq}n''.\ \|c\|_{t\ i})$› **have** $\langle c \wedge t\rangle \leq n''$ **using** *assms*(*1*) *lActive-less* **by** *auto*
with ‹$n'' < {}_c{\uparrow}_t x$› **have** $_c{\downarrow}_t(n'')$<x **using** *p2c-mono-c2p-strict* **by** *simp*
ultimately have γ' (*lnth* ((π_c(*inf-llist* t)) @$_l$ (*inf-llist* t'))) ($_c{\downarrow}_t(n'')$)
using *a1* **by** *auto*
with ‹$\exists\, i{\geq}n.\ \|c\|_{t\ i}$› ‹$\neg(\exists\, i{\geq}n''.\ \|c\|_{t\ i})$› **show** *?thesis* **using** *validCI-cont* **by** *blast*
qed
qed
qed
ultimately show *?thesis* **using** ‹$\neg(\exists\, i{\geq}{}_c{\uparrow}_t(x).\ \|c\|_{t\ i})$› **by** *blast*
next
assume $\neg(x \geq$ *llength* (π_c(*inf-llist* t)))
hence x<*llength* (π_c(*inf-llist* t)) **by** *simp*
then obtain n'::*nat* **where** $x{=}\langle c \#_{n'}$ *inf-llist* $t\rangle$ **using** *nAct-exists* **by** *blast*
with ‹*enat* $x <$ *llength* (π_c(*inf-llist* t))› **have** $\exists\, i{\geq}n'.\ \|c\|_{t\ i}$ **using** *nAct-less-llength-active* **by** *force*
then obtain i **where** $i{\geq}n'$ **and** $\|c\|_{t\ i}$ **and** $\neg\ (\exists\, k{\geq}n'.\ k < i \wedge \|c\|_{t\ k})$ **using** *nact-exists* **by** *blast*
moreover have ($\forall\, n''{\geq}\ \langle c \Leftarrow t\rangle_i.\ n''{\leq}\langle c \rightarrow t\rangle_i \longrightarrow$ *eval* $c\ t\ t'\ n''\ \gamma$)
proof
fix n'' **show** $\langle c \Leftarrow t\rangle_i \leq n'' \longrightarrow n'' \leq \langle c \rightarrow t\rangle_i \longrightarrow$ *eval* $c\ t\ t'\ n''\ \gamma$
proof(*rule HOL.impI*[*OF HOL.impI*])
assume $\langle c \Leftarrow t\rangle_i \leq n''$ **and** $n'' \leq \langle c \rightarrow t\rangle_i$
hence *the-enat* ($\langle c \#_{enat\ i}$ *inf-llist* $t\rangle$) = *the-enat* ($\langle c \#_{enat\ n''}$ *inf-llist* $t\rangle$)
using *nAct-same* **by** *simp*
moreover from ‹$\|c\|_{t\ i}$› **have** $\|c\|_{t\ \langle c \rightarrow t\rangle_i}$ **using** *nxtActI* **by** *auto*
with ‹$n'' \leq \langle c \rightarrow t\rangle_i$› **have** $\exists\, i{\geq}n''.\ \|c\|_{t\ i}$ **using** *dual-order.strict-implies-order* **by** *auto*
moreover have γ (*lnth* ((π_c(*inf-llist* t)) @$_l$ (*inf-llist* t'))) (*the-enat* ($\langle c \#_{enat\ i}$ *inf-llist* $t\rangle$))
proof −
have *enat* i − *1* < *llength* (*inf-llist* t) **by** (*simp add*: *one-enat-def*)
with ‹$x{=}\langle c \#_{n'}$ *inf-llist* $t\rangle$› ‹$i{\geq}n'$› ‹$\neg\ (\exists\, k{\geq}n'.\ k < i \wedge \|c\|_{t\ k})$› **have** $x{=}\langle c \#_i$ *inf-llist* $t\rangle$
using *one-enat-def nAct-not-active-same* **by** *simp*
moreover have $\langle c \#_i$ *inf-llist* $t\rangle{\neq}\infty$ **by** *simp*
ultimately have x=*the-enat*($\langle c \#_i$ *inf-llist* $t\rangle$) **by** *fastforce*
thus *?thesis* **using** ‹γ (*lnth* ((π_c(*inf-llist* t)) @$_l$ (*inf-llist* t'))) x› **by** *blast*
qed
with ‹*the-enat* ($\langle c \#_{enat\ i}$ *inf-llist* $t\rangle$) = *the-enat* ($\langle c \#_{enat\ n''}$ *inf-llist* $t\rangle$)› **have**
γ (*lnth* ((π_c(*inf-llist* t)) @$_l$ (*inf-llist* t'))) (*the-enat* ($\langle c \#_{enat\ n''}$ *inf-llist* $t\rangle$)) **by** *simp*
ultimately show *eval* $c\ t\ t'\ n''\ \gamma$ **using** *validCI-act* **by** *blast*
qed
qed
moreover have $i{\geq}\langle c \rightarrow t\rangle_n$
proof −
have *enat* i − *1* < *llength* (*inf-llist* t) **by** (*simp add*: *one-enat-def*)
with ‹$x{=}\langle c \#_{n'}$ *inf-llist* $t\rangle$› ‹$i{\geq}n'$› ‹$\neg\ (\exists\, k{\geq}n'.\ k < i \wedge \|c\|_{t\ k})$› **have** $x{=}\langle c \#_i$ *inf-llist* $t\rangle$
using *one-enat-def nAct-not-active-same* **by** *simp*
moreover have $\langle c \#_i$ *inf-llist* $t\rangle{\neq}\infty$ **by** *simp*
ultimately have x=*the-enat*($\langle c \#_i$ *inf-llist* $t\rangle$) **by** *fastforce*
with ‹$x{\geq}$*the-enat* ($\langle c \#_n$ *inf-llist* $t\rangle$)›
have *the-enat* ($\langle c \#_i$ *inf-llist* $t\rangle$)$\geq$*the-enat* ($\langle c \#_n$ *inf-llist* $t\rangle$) **by** *simp*
with ‹$\|c\|_{t\ i}$› **show** *?thesis* **using** *active-geq-nxtAct* **by** *simp*
qed

```
    moreover have ∀ n''≥⟨c ⇐ t⟩n. n'' < ⟨c ⇐ t⟩i ⟶ eval c t t' n'' γ'
    proof
      fix n'' show ⟨c ⇐ t⟩n ≤ n'' ⟶ n'' < ⟨c ⇐ t⟩i ⟶ eval c t t' n'' γ'
      proof (rule HOL.impI[OF HOL.impI])
        assume ⟨c ⇐ t⟩n ≤ n'' and n'' < ⟨c ⇐ t⟩i
        moreover have ⟨c ⇐ t⟩i≤i by simp
        ultimately have ∃ i≥n''. ‖c‖t i using ‹‖c‖t i› by (meson less-le less-le-trans)
        with ‹n''≥⟨c ⇐ t⟩n› have the-enat (⟨c #n'' inf-llist t⟩) ≥ the-enat (⟨c #n inf-llist t⟩)
          using nAct-mono-lNact ‹∃ i≥n. ‖c‖t i› by simp
        moreover have the-enat (⟨c #n'' inf-llist t⟩) < x
        proof −
          from ‹n'' < ⟨c ⇐ t⟩i› ‹⟨c ⇐ t⟩i ≤ i› have n'' < i using dual-order.strict-trans1 by arith
          with ‹n'' < ⟨c ⇐ t⟩i› have ∃ i'≥n''. i' < i ∧ ‖c‖t i' using lNact-least[of i n''] by fastforce
          hence ⟨c #n'' inf-llist t⟩ < ⟨c #i inf-llist t⟩ using nAct-less by auto
          moreover have enat i − 1 < llength (inf-llist t) by (simp add: one-enat-def)
          with ‹x=⟨c #n' inf-llist t⟩› ‹i≥n'› ‹¬ (∃ k≥n'. k < i ∧ ‖c‖t k)› have x=⟨c #i inf-llist t⟩
            using one-enat-def nAct-not-active-same by simp
          moreover have ⟨c #n'' inf-llist t⟩≠∞ by simp
          ultimately show ?thesis by (metis enat-ord-simps(2) enat-the-enat)
        qed
        ultimately have γ' (lnth ((πc(inf-llist t)) @l (inf-llist t'))) (the-enat (⟨c #n'' inf-llist t⟩))
          using a1 by auto
        with ‹∃ i≥n''. ‖c‖t i› show eval c t t' n'' γ' using validCI-act by blast
      qed
    qed
    ultimately show ?thesis using ‹‖c‖t i› by auto
  qed
qed

lemma untilEN[elim]:
  fixes n::nat
    and n'::nat
    and t::nat ⇒ cnf
    and t'::nat ⇒ 'cmp
    and c::'id
  assumes ∄ i. i≥n ∧ ‖c‖t i
    and eval c t t' n (γ' 𝔘b γ)
  shows ∃ n'≥n. eval c t t' n' γ ∧
    (∀ n''≥n. n'' < n' ⟶ eval c t t' n'' γ')
proof cases
  assume ∃ i. ‖c‖t i
  moreover from ‹eval c t t' n (γ' 𝔘b γ)›
    have eval c t t' n (λ t n. ∃ n''≥n. γ t n'' ∧ (∀ n'≥n. n' < n'' ⟶ γ' t n')) using until-def by simp
  ultimately have ∃ n''≥c↓t(n). γ (lnth (πcinf-llist t @l inf-llist t')) n'' ∧
    (∀ n'≥c↓t(n). n' < n'' ⟶ γ' (lnth (πcinf-llist t @l inf-llist t')) n')
    using validCE-cont[where γ=λ t n. ∃ n''≥n. γ t n'' ∧ (∀ n'≥n. n' < n'' ⟶ γ' t n')]
    ‹∄ i. i≥n ∧ ‖c‖t i› by blast
  then obtain x where x≥c↓t(n) and γ (lnth ((πc(inf-llist t)) @l (inf-llist t'))) x
    and ∀ x'≥c↓t(n). x'<x ⟶ γ' (lnth ((πc(inf-llist t)) @l (inf-llist t'))) x' by auto
  moreover from ‹¬(∃ i≥n. ‖c‖t i)› have the-enat (llength (πc(inf-llist t))) − 1 < x
  proof −
    have ⟨c ∧ t⟩ < n
    proof (rule ccontr)
      assume ¬⟨c ∧ t⟩ < n
      hence ⟨c ∧ t⟩ ≥ n by simp
```

moreover from ‹$\exists\, i.\ \|c\|_{t\ i}$› ‹$\neg\ (\exists\, i{\geq}n.\ \|c\|_{t\ i})$› **have** $\|c\|_{t\ \langle c\ \wedge\ t\rangle}$
using *lActive-active less-or-eq-imp-le* **by** *blast*
ultimately show *False* **using** ‹$\neg\ (\exists\, i{\geq}n.\ \|c\|_{t\ i})$› **by** *simp*
qed
hence *the-enat* (*llength* (π_c(*inf-llist t*))) − *1* < $_c{\downarrow_t}(n)$ **using** *cnf2bhv-greater-llength* **by** *simp*
with ‹$x{\geq}_c{\downarrow_t}(n)$› **show** *?thesis* **by** *simp*
qed
hence $x = {}_c{\downarrow_t}({}_c{\uparrow_t}(x))$ **using** *cnf2bhv-bhv2cnf* **by** *simp*
ultimately have γ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*))) ($_c{\downarrow_t}({}_c{\uparrow_t}(x))$) **by** *simp*
moreover from ‹$\neg(\exists\, i{\geq}n.\ \|c\|_{t\ i})$› **have** $\neg(\exists\, i{\geq}{}_c{\uparrow_t}(x).\ \|c\|_{t\ i})$
proof −
from ‹$\neg(\exists\, i{\geq}n.\ \|c\|_{t\ i})$› **have** *lfinite* ($\pi_c$(*inf-llist t*)) **using** *proj-finite2* **by** *simp*
then obtain *z* **where** $\forall\, n''{>}z.\ \neg\ \|c\|_{t\ n''}$ **using** *proj-finite-bound* **by** *blast*
moreover from ‹*the-enat* (*llength* (π_c(*inf-llist t*))) − *1* < *x*› **have** $\langle c\ \wedge\ t\rangle$ < $_c{\uparrow_t}(x)$
using *bhv2cnf-greater-lActive* **by** *simp*
ultimately show *?thesis* **using** *lActive-greater-active-all* **by** *simp*
qed
ultimately have *eval c t t'* ($_c{\uparrow_t}(x)$) γ **using** *validCI-cont* ‹$\exists\, i.\ \|c\|_{t\ i}$› **by** *blast*
moreover from ‹$\exists\, i.\ \|c\|_{t\ i}$› ‹$\neg(\exists\, i{\geq}n.\ \|c\|_{t\ i})$› **have** $\langle c\ \wedge\ t\rangle \leq n$ **using** *lActive-less*[*of c t - n*] **by** *auto*
with ‹$x{\geq}_c{\downarrow_t}(n)$› **have** $n \leq {}_c{\uparrow_t}(x)$ **using** *p2c-mono-c2p* **by** *blast*
moreover have $\forall\, n''{\geq}n.\ n'' < {}_c{\uparrow_t}(x) \longrightarrow$ *eval c t t' n'' γ'*
proof (*rule HOL.allI*[*OF HOL.impI*[*OF HOL.impI*]])
fix *n''* **assume** $n \leq n''$ **and** $n'' < {}_c{\uparrow_t}(x)$
hence $_c{\downarrow_t}(n''){\geq}{}_c{\downarrow_t}(n)$ **using** *cnf2bhv-mono* **by** *simp*
moreover have $n''{<}{}_c{\uparrow_t}(x)$ **by** (*simp add*: ‹$n'' < {}_c{\uparrow_t}x$›)
with ‹$\langle c\ \wedge\ t\rangle \leq n$› ‹$n \leq n''$› **have** $_c{\downarrow_t}(n''){<}{}_c{\downarrow_t}({}_c{\uparrow_t}(x))$ **using** *cnf2bhv-mono-strict* **by** *simp*
with ‹$x = {}_c{\downarrow_t}({}_c{\uparrow_t}(x))$› **have** $_c{\downarrow_t}(n''){<}\ x$ **by** *simp*
ultimately have γ' (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*))) ($_c{\downarrow_t}(n'')$)
using ‹$\forall\, x'{\geq}{}_c{\downarrow_t}(n).\ x'{<}x \longrightarrow \gamma'$ (*lnth* ((π_c(*inf-llist t*)) @$_l$ (*inf-llist t'*))) x'› **by** *simp*
moreover from ‹$n \leq n''$› **have** $\nexists\, i.\ i{\geq}n'' \wedge \|c\|_{t\ i}$ **using** ‹$\nexists\, i.\ i{\geq}n \wedge \|c\|_{t\ i}$› **by** *simp*
ultimately show *eval c t t' n'' γ'* **using** *validCI-cont* **using** ‹$\exists\, i.\ \|c\|_{t\ i}$› **by** *blast*
qed
ultimately show *?thesis* **by** *auto*
next
assume $\neg(\exists\, i.\ \|c\|_{t\ i})$
moreover from ‹*eval c t t' n* (γ' $\mathfrak{U}_b$ γ)›
have *eval c t t' n* (λ *t n*. $\exists\, n''{\geq}n.\ \gamma$ *t n''* $\wedge$ ($\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma'$ *t n'*)) **using** *until-def* **by** *simp*
ultimately have $\exists\, n''{\geq}n.\ \gamma$ (*lnth* (π_c*inf-llist t* @$_l$ *inf-llist t'*)) *n''*
$\wedge$ ($\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma'$ (*lnth* (π_c*inf-llist t* @$_l$ *inf-llist t'*)) *n'*) **using** ‹$\neg(\exists\, i.\ \|c\|_{t\ i})$›
validCE-not-act[**where** $\gamma{=}\lambda$ *t n*. $\exists\, n''{\geq}n.\ \gamma$ *t n''* $\wedge$ ($\forall\, n'{\geq}n.\ n' < n'' \longrightarrow \gamma'$ *t n'*)] **by** *blast*
with ‹$\neg(\exists\, i.\ \|c\|_{t\ i})$› **show** *?thesis* **using** *validCI-not-act* **by** *blast*
qed

2.4.13 Weak Until

definition *wuntil* :: (*'cmp bta*) ⇒ (*'cmp bta*) ⇒ (*'cmp bta*) (**infixl** $\mathfrak{W}_b$ *20*)
where $\gamma'\ \mathfrak{W}_b\ \gamma \equiv \gamma'\ \mathfrak{U}_b\ \gamma\ \vee^b\ \Box_b(\gamma')$

end

end

References

[1] A. Lochbihler. Coinduction. *The Archive of Formal Proof s. https://isa-afp.org/entries/Coinductive.shtml*, 2010.

[2] D. Marmsoler. On the semantics of temporal specifications of component-behavior for dynamic architectures. In *Eleventh International Symposium on Theoretical Aspects of Software Engineering*. Springer, 2017.

[3] D. Marmsoler. Towards a calculus for dynamic architectures. In *International Colloquium on Theoretical Aspects of Computing*. Springer, 2017.

[4] D. Marmsoler and M. Gleirscher. On activation, connection, and behavior in dynamic architectures. *Scientific Annals of Computer Science*, 26(2):187–248, 2016.

[5] D. Marmsoler and M. Gleirscher. Specifying properties of dynamic architectures using configuration traces. In *International Colloquium on Theoretical Aspects of Computing*, pages 235–254. Springer, 2016.