

A shallow embedding of HyperCTL*

Markus N. Rabe Peter Lammich Andrei Popescu

Contents

1	Introduction	1
2	Preliminaries	1
3	Shallow embedding of HyperCTL*	2
3.1	Kripke structures and paths	2
3.2	Shallow representations of formulas	3
3.3	Reasoning rules	5
3.4	More derived operators	9
4	Noninterference à la Goguen and Meseguer	10
4.1	Goguen-Meseguer noninterference	10
4.2	Specialized Kripke structures	10
4.3	Faithful representation as a HyperCTL* property	12
5	Deep representation of HyperCTL* – syntax and semantics	19
5.1	Path variables and environments	19
5.2	The semantic operator	21
5.3	The conjunction of a finite set of formulas	24
6	Noninterference for models with finitely many users, commands and outputs	25

1 Introduction

We formalize HyperCTL*, a temporal logic for expressing security properties introduced in [1,2]. We first define a shallow embedding of HyperCTL*, within which we prove inductive and coinductive rules for the operators. Then we show that a HyperCTL* formula captures Goguen-Meseguer noninterference, a landmark information flow property. We also define a deep embedding and connect it to the shallow embedding by a denotational semantics, for which we prove sanity w.r.t. dependence on the free variables. Finally, we show that under some finiteness assumptions about the model, noninterference is given by a (finitary) syntactic formula.

For the semantics of HyperCTL*, we mainly follow the earlier paper [1]. The Kripke structure for representing noninterference is essentially that of [1, Appendix B] – however, instead of using the formula from [1, Appendix B], we further add idle transitions to the Kripke structure and use the simpler formula from [1, Section 2.4].

[1] Bernd Finkbeiner, Markus N. Rabe and César Sánchez. A Temporal Logic for Hyperproperties. CoRR, abs/1306.6657, 2013.

[2] Michael R. Clarkson, Bernd Finkbeiner, Masoud Kolehini, Kristopher K. Micinski, Markus N. Rabe and César Sánchez. Temporal Logics for Hyperproperties. POST 2014, 265-284.

2 Preliminaries

abbreviation *any where any* $\equiv$ *undefined*

lemma *append-singl-rev*: $a \# as = [a] @ as \langle proof \rangle$

lemma *list-pair-induct*[*case-names Nil Cons*]:
assumes $P []$ **and** $\bigwedge a b list. P list \implies P ((a,b) \# list)$
shows $P lista$
 $\langle proof \rangle$

lemma *list-pair-case*[*elim, case-names Nil Cons*]:
assumes $xs = [] \implies P$ **and** $\bigwedge a b list. xs = (a,b) \# list \implies P$
shows P
 $\langle proof \rangle$

definition *asList* :: $'a \text{ set} \Rightarrow 'a \text{ list}$ **where**
 $asList A \equiv SOME as. distinct as \wedge set as = A$

lemma *asList*:
assumes *finite A* **shows** $distinct (asList A) \wedge set (asList A) = A$
 $\langle proof \rangle$

lemmas *distinct-asList* = *asList*[*THEN conjunct1*]
lemmas *set-asList* = *asList*[*THEN conjunct2*]

lemma *map-sdrop*[*simp*]: $sdrop\ 0 = id$
 $\langle proof \rangle$

lemma *stl-o-sdrop*[*simp*]: $stl\ o\ sdrop\ n = sdrop\ (Suc\ n)$
 $\langle proof \rangle$

lemma *sdrop-o-stl*[*simp*]: $sdrop\ n\ o\ stl = sdrop\ (Suc\ n)$
 $\langle proof \rangle$

lemma *hd-stake[simp]*: $i > 0 \implies \text{hd } (\text{stake } i \ \pi) = \text{shd } \pi$
 <proof>

3 Shallow embedding of HyperCTL*

We define a notion of “shallow” HyperCTL* formula (sfmla) that captures HyperCTL* binders as meta-level HOL binders. We also define a proof system for this shallow embedding.

3.1 Kripke structures and paths

type-synonym $(\text{'state}, \text{'aprop}) \text{ path} = (\text{'state} \times \text{'aprop set}) \text{ stream}$

abbreviation *stateOf* **where** $\text{stateOf } \pi \equiv \text{fst } (\text{shd } \pi)$

abbreviation *apropsOf* **where** $\text{apropsOf } \pi \equiv \text{snd } (\text{shd } \pi)$

locale *Kripke* =

fixes $S :: \text{'state set}$ **and** $s0 :: \text{'state}$ **and** $\delta :: \text{'state} \Rightarrow \text{'state set}$

and $AP :: \text{'aprop set}$ **and** $L :: \text{'state} \Rightarrow \text{'aprop set}$

assumes $s0: s0 \in S$ **and** $\delta: \bigwedge s. s \in S \implies \delta s \subseteq S$

and $L: \bigwedge s. s \in S \implies L s \subseteq AP$

begin

Well-formed paths

coinductive *wfp* :: $\text{'aprop set} \Rightarrow (\text{'state}, \text{'aprop}) \text{ path} \Rightarrow \text{bool}$

for $AP' :: \text{'aprop set}$

where

intro:

$\llbracket s \in S; A \subseteq AP'; A \cap AP = L s; \text{stateOf } \pi \in \delta s; \text{wfp } AP' \ \pi \rrbracket$

$\implies$

$\text{wfp } AP' ((s, A) \#\# \pi)$

lemma *wfp*:

$\text{wfp } AP' \ \pi \longleftrightarrow$

$(\forall i. \text{fst } (\pi !! i) \in S \wedge \text{snd } (\pi !! i) \subseteq AP' \wedge$

$\text{snd } (\pi !! i) \cap AP = L (\text{fst } (\pi !! i)) \wedge$

$\text{fst } (\pi !! (\text{Suc } i)) \in \delta (\text{fst } (\pi !! i))$

)

(**is** ?L $\longleftrightarrow (\forall i. ?R i)$)

<proof>

lemma *wfp-sdrop[simp]*:

$\text{wfp } AP' \ \pi \implies \text{wfp } AP' (\text{sdrop } i \ \pi)$

<proof>

end-of-context Kripke

3.2 Shallow representations of formulas

A shallow (representation of a) HyperCTL* formula will be a predicate on lists of paths. The atomic formulas (operator *atom*) are parameterized by atomic propositions (as customary in temporal logic), and additionally by a number indicating the position, in the list of paths, of the path to which the atomic proposition refers – for example, *atom a i* holds for the list of paths πl just in case proposition *a* holds at the first state of $\pi l[i]$, the *i*'th path in πl . The temporal operators *next* and *until* act on all the paths of the argument list πl synchronously. Finally, the existential quantifier refers to the existence of a path whose origin state is that of the last path in πl .

As an example: *exi (exi (until (atom a 0) (atom b 1)))* holds for the empty list iff there exist two paths ρ_0 and ρ_1 such that, synchronously, *a* holds on ρ_0 until *b* holds on ρ_1 . Another example will be the formula encoding Goguen-Meseguer noninterference.

Shallow HyperCTL* formulas:

type-synonym ('state,'aprop) sfmla = ('state,'aprop) path list $\Rightarrow$ bool

locale Shallow = Kripke S s0 δ AP L
for S :: 'state set **and** s0 :: 'state **and** δ :: 'state $\Rightarrow$ 'state set
and AP :: 'aprop set **and** L :: 'state $\Rightarrow$ 'aprop set
+
fixes AP' **assumes** AP-AP': AP $\subseteq$ AP'
begin

Primitive operators

definition fls :: ('state,'aprop) sfmla **where**
fls $\pi l \equiv$ False

definition atom :: 'aprop $\Rightarrow$ nat $\Rightarrow$ ('state,'aprop) sfmla **where**
atom a i $\pi l \equiv a \in \text{apropsOf } (\pi l[i])$

definition neg :: ('state,'aprop) sfmla $\Rightarrow$ ('state,'aprop) sfmla **where**
neg $\varphi \pi l \equiv \neg \varphi \pi l$

definition dis :: ('state,'aprop) sfmla $\Rightarrow$ ('state,'aprop) sfmla $\Rightarrow$ ('state,'aprop) sfmla **where**
dis $\varphi \psi \pi l \equiv \varphi \pi l \vee \psi \pi l$

definition next :: ('state,'aprop) sfmla $\Rightarrow$ ('state,'aprop) sfmla **where**
next $\varphi \pi l \equiv \varphi (\text{map stl } \pi l)$

definition until :: ('state,'aprop) sfmla $\Rightarrow$ ('state,'aprop) sfmla $\Rightarrow$ ('state,'aprop) sfmla **where**
until $\varphi \psi \pi l \equiv$
 $\exists i. \psi (\text{map } (\text{sdrop } i) \pi l) \wedge (\forall j \in \{0..<i\}. \varphi (\text{map } (\text{sdrop } j) \pi l))$

definition exii :: ('state,'aprop) sfmla $\Rightarrow$ ('state,'aprop) sfmla **where**
exii $\varphi \pi l \equiv$
 $\exists \pi. \text{wfp } AP' \pi \wedge \text{stateOf } \pi = (\text{if } \pi l \neq [] \text{ then stateOf } (\text{last } \pi l) \text{ else } s0)$
 $\wedge \varphi (\pi l @ [\pi])$

definition $exi :: (('state, 'aprop) path \Rightarrow ('state, 'aprop) sfmla) \Rightarrow ('state, 'aprop) sfmla$ **where**
 $exi F \pi l \equiv$
 $\exists \pi. wfp AP' \pi \wedge stateOf \pi = (if \pi l \neq [] then stateOf (last \pi l) else s0)$
 $\wedge F \pi \pi l$

Derived operators

definition $tr \equiv neg\ fls$
definition $con\ \varphi\ \psi \equiv neg\ (dis\ (neg\ \varphi)\ (neg\ \psi))$
definition $imp\ \varphi\ \psi \equiv dis\ (neg\ \varphi)\ \psi$
definition $eq\ \varphi\ \psi \equiv con\ (imp\ \varphi\ \psi)\ (imp\ \psi\ \varphi)$
definition $fall\ F \equiv neg\ (exi\ (\lambda \pi. neg\ (F\ \pi)))$
definition $ev\ \varphi \equiv until\ tr\ \varphi$
definition $alw\ \varphi \equiv neg\ (ev\ (neg\ \varphi))$
definition $wuntil\ \varphi\ \psi \equiv dis\ (until\ \varphi\ \psi)\ (alw\ \varphi)$

lemmas $main-op-defs =$
 $fls-def\ atom-def\ neg-def\ dis-def\ next-def\ until-def\ exi-def$

lemmas $der-op-defs =$
 $tr-def\ con-def\ imp-def\ eq-def\ ev-def\ alw-def\ wuntil-def\ fall-def$

lemmas $op-defs = main-op-defs\ der-op-defs$

3.3 Reasoning rules

We provide introduction, elimination, unfolding and (co)induction rules for the connectives and quantifiers.

Boolean operators

lemma $fls-elim[elim!]$:
assumes $fls\ \pi l$ **shows** φ
 $\langle proof \rangle$

lemma $tr-intro[intro!, simp]$: $tr\ \pi l$
 $\langle proof \rangle$

lemma $dis-introL[intro]$:
assumes $\varphi\ \pi l$ **shows** $dis\ \varphi\ \psi\ \pi l$
 $\langle proof \rangle$

lemma $dis-introR[intro]$:
assumes $\psi\ \pi l$ **shows** $dis\ \varphi\ \psi\ \pi l$
 $\langle proof \rangle$

lemma $dis-elim[elim]$:
assumes $dis\ \varphi\ \psi\ \pi l$ **and** $\varphi\ \pi l \implies \chi$ **and** $\psi\ \pi l \implies \chi$
shows χ
 $\langle proof \rangle$

lemma *con-intro*[*intro!*]:
assumes $\varphi \ \pi l$ **and** $\psi \ \pi l$ **shows** $\text{con } \varphi \ \psi \ \pi l$
 $\langle \text{proof} \rangle$

lemma *con-elim*[*elim*]:
assumes $\text{con } \varphi \ \psi \ \pi l$ **and** $\varphi \ \pi l \implies \psi \ \pi l \implies \chi$ **shows** χ
 $\langle \text{proof} \rangle$

lemma *neg-intro*[*intro!*]:
assumes $\varphi \ \pi l \implies \text{False}$ **shows** $\text{neg } \varphi \ \pi l$
 $\langle \text{proof} \rangle$

lemma *neg-elim*[*elim*]:
assumes $\text{neg } \varphi \ \pi l$ **and** $\varphi \ \pi l$ **shows** χ
 $\langle \text{proof} \rangle$

lemma *imp-intro*[*intro!*]:
assumes $\varphi \ \pi l \implies \psi \ \pi l$ **shows** $\text{imp } \varphi \ \psi \ \pi l$
 $\langle \text{proof} \rangle$

lemma *imp-elim*[*elim*]:
assumes $\text{imp } \varphi \ \psi \ \pi l$ **and** $\varphi \ \pi l$ **and** $\psi \ \pi l \implies \chi$ **shows** χ
 $\langle \text{proof} \rangle$

lemma *imp-mp*[*elim*]:
assumes $\text{imp } \varphi \ \psi \ \pi l$ **and** $\varphi \ \pi l$ **shows** $\psi \ \pi l$
 $\langle \text{proof} \rangle$

lemma *eq-intro*[*intro!*]:
assumes $\varphi \ \pi l \implies \psi \ \pi l$ **and** $\psi \ \pi l \implies \varphi \ \pi l$ **shows** $\text{eq } \varphi \ \psi \ \pi l$
 $\langle \text{proof} \rangle$

lemma *eq-elimL*[*elim*]:
assumes $\text{eq } \varphi \ \psi \ \pi l$ **and** $\varphi \ \pi l$ **and** $\psi \ \pi l \implies \chi$ **shows** χ
 $\langle \text{proof} \rangle$

lemma *eq-elimR*[*elim*]:
assumes $\text{eq } \varphi \ \psi \ \pi l$ **and** $\psi \ \pi l$ **and** $\varphi \ \pi l \implies \chi$ **shows** χ
 $\langle \text{proof} \rangle$

lemma *eq-equals*: $\text{eq } \varphi \ \psi \ \pi l \longleftrightarrow \varphi \ \pi l = \psi \ \pi l$
 $\langle \text{proof} \rangle$

Quantifiers

lemma *exi-intro*[*intro*]:
assumes $\text{wfp } AP' \ \pi$
and $\pi l \neq [] \implies \text{stateOf } \pi = \text{stateOf } (\text{last } \pi l)$
and $\pi l = [] \implies \text{stateOf } \pi = s0$

and $F \pi \pi l$
shows $exi F \pi l$
 $\langle proof \rangle$

lemma *exi-elim*[*elim*]:
assumes $exi F \pi l$
and
 $\bigwedge \pi. \llbracket wfp AP' \pi; \pi l \neq [] \implies stateOf \pi = stateOf (last \pi l); \pi l = [] \implies stateOf \pi = s0; F \pi \pi l \rrbracket \implies \chi$
shows χ
 $\langle proof \rangle$

lemma *fall-intro*[*intro*]:
assumes
 $\bigwedge \pi. \llbracket wfp AP' \pi; \pi l \neq [] \implies stateOf \pi = stateOf (last \pi l) ; \pi l = [] \implies stateOf \pi = s0 \rrbracket$
 $\implies F \pi \pi l$
shows $fall F \pi l$
 $\langle proof \rangle$

lemma *fall-elim*[*elim*]:
assumes $fall F \pi l$
and
 $(\bigwedge \pi. \llbracket wfp AP' \pi; \pi l \neq [] \implies stateOf \pi = stateOf (last \pi l); \pi l = [] \implies stateOf \pi = s0 \rrbracket$
 $\implies F \pi \pi l)$
 $\implies \chi$
shows χ
 $\langle proof \rangle$

Temporal connectives

lemma *next-intro*[*intro*]:
assumes $\varphi (map stl \pi l)$ **shows** $next \varphi \pi l$
 $\langle proof \rangle$

lemma *next-elim*[*elim*]:
assumes $next \varphi \pi l$ **and** $\varphi (map stl \pi l) \implies \chi$ **shows** χ
 $\langle proof \rangle$

lemma *until-introR*[*intro*]:
assumes $\psi \pi l$ **shows** $until \varphi \psi \pi l$
 $\langle proof \rangle$

lemma *until-introL*[*intro*]:
assumes $\varphi: \varphi \pi l$ **and** $u: until \varphi \psi (map stl \pi l)$
shows $until \varphi \psi \pi l$
 $\langle proof \rangle$

The elimination rules for until and eventually are induction rules.

lemma *until-induct*[*induct pred: until, consumes 1, case-names Base Step*]:
assumes $u: until \varphi \psi \pi l$
and $b: \bigwedge \pi l. \psi \pi l \implies \chi \pi l$

and $i: \bigwedge \pi l. \llbracket \varphi \pi l; \text{until } \varphi \psi (\text{map stl } \pi l); \chi (\text{map stl } \pi l) \rrbracket \implies \chi \pi l$
shows $\chi \pi l$
 $\langle \text{proof} \rangle$

lemma *until-unfold*:
 $\text{until } \varphi \psi \pi l = (\psi \pi l \vee \varphi \pi l \wedge \text{until } \varphi \psi (\text{map stl } \pi l))$ (**is** $?L = ?R$)
 $\langle \text{proof} \rangle$

lemma *ev-introR[intro]*:
assumes $\varphi \pi l$ **shows** $\text{ev } \varphi \pi l$
 $\langle \text{proof} \rangle$

lemma *ev-introL[intro]*:
assumes $\text{ev } \varphi (\text{map stl } \pi l)$ **shows** $\text{ev } \varphi \pi l$
 $\langle \text{proof} \rangle$

lemma *ev-induct[induct pred: ev, consumes 1, case-names Base Step]*:
assumes $\text{ev } \varphi \pi l$ **and** $\bigwedge \pi l. \varphi \pi l \implies \chi \pi l$
and $\bigwedge \pi l. \llbracket \text{ev } \varphi (\text{map stl } \pi l); \chi (\text{map stl } \pi l) \rrbracket \implies \chi \pi l$
shows $\chi \pi l$
 $\langle \text{proof} \rangle$

lemma *ev-unfold*:
 $\text{ev } \varphi \pi l = (\varphi \pi l \vee \text{ev } \varphi (\text{map stl } \pi l))$
 $\langle \text{proof} \rangle$

lemma *ev*: $\text{ev } \varphi \pi l \longleftrightarrow (\exists i. \varphi (\text{map } (\text{sdrop } i) \pi l))$
 $\langle \text{proof} \rangle$

The introduction rules for always and weak until are coinduction rules.

lemma *alw-coinduct[coinduct pred: alw, consumes 1, case-names Hyp]*:
assumes $\chi \pi l$
and $\bigwedge \pi l. \chi \pi l \implies \text{alw } \varphi \pi l \vee (\varphi \pi l \wedge \chi (\text{map stl } \pi l))$
shows $\text{alw } \varphi \pi l$
 $\langle \text{proof} \rangle$

lemma *alw-elim[elim]*:
assumes $\text{alw } \varphi \pi l$
and $\llbracket \varphi \pi l; \text{alw } \varphi (\text{map stl } \pi l) \rrbracket \implies \chi$
shows χ
 $\langle \text{proof} \rangle$

lemma *alw-destL*: $\text{alw } \varphi \pi l \implies \varphi \pi l$ $\langle \text{proof} \rangle$
lemma *alw-destR*: $\text{alw } \varphi \pi l \implies \text{alw } \varphi (\text{map stl } \pi l)$ $\langle \text{proof} \rangle$

lemma *alw-unfold*:
 $\text{alw } \varphi \pi l = (\varphi \pi l \wedge \text{alw } \varphi (\text{map stl } \pi l))$
 $\langle \text{proof} \rangle$

lemma *alw*: $alw\ \varphi\ \pi l \longleftrightarrow (\forall\ i.\ \varphi\ (map\ (sdrop\ i)\ \pi l))$
 $\langle proof \rangle$

lemma *sdrop-imp-alw*:
assumes $\bigwedge i.\ (\bigwedge j.\ j \leq i \implies \varphi\ [sdrop\ j\ \pi,\ sdrop\ j\ \pi']) \implies \psi\ [sdrop\ i\ \pi,\ sdrop\ i\ \pi']$
shows $imp\ (alw\ \varphi)\ (alw\ \psi)\ [\pi,\ \pi']$
 $\langle proof \rangle$

lemma *wuntil-coinduct*[*coinduct pred: wuntil, consumes 1, case-names Hyp*]:
assumes $\chi: \chi\ \pi l$
and $0: \bigwedge \pi l.\ \chi\ \pi l \implies \psi\ \pi l \vee (\varphi\ \pi l \wedge \chi\ (map\ stl\ \pi l))$
shows $wuntil\ \varphi\ \psi\ \pi l$
 $\langle proof \rangle$

lemma *wuntil-elim*[*elim*]:
assumes $w: wuntil\ \varphi\ \psi\ \pi l$
and $1: \psi\ \pi l \implies \chi$
and $2: \llbracket \varphi\ \pi l; wuntil\ \varphi\ \psi\ (map\ stl\ \pi l) \rrbracket \implies \chi$
shows χ
 $\langle proof \rangle$

lemma *wuntil-unfold*:
 $wuntil\ \varphi\ \psi\ \pi l = (\psi\ \pi l \vee \varphi\ \pi l \wedge wuntil\ \varphi\ \psi\ (map\ stl\ \pi l))$
 $\langle proof \rangle$

3.4 More derived operators

The conjunction of an arbitrary set of formulas:

definition *scon* ::
 $('state, 'aprop)\ sfmla\ set \Rightarrow ('state, 'aprop)\ sfmla$ **where**
 $scon\ \varphi s\ \pi l \equiv \forall\ \varphi \in \varphi s.\ \varphi\ \pi l$

lemma *mcon-intro*[*intro*]:
assumes $\bigwedge \varphi.\ \varphi \in \varphi s \implies \varphi\ \pi l$ **shows** $scon\ \varphi s\ \pi l$
 $\langle proof \rangle$

lemma *scon-elim*[*elim*]:
assumes $scon\ \varphi s\ \pi l$ **and** $(\bigwedge \varphi.\ \varphi \in \varphi s \implies \varphi\ \pi l) \implies \chi$
shows χ
 $\langle proof \rangle$

Double-binding forall:

definition *fall2* $F \equiv fall\ (\lambda\ \pi.\ fall\ (F\ \pi))$

lemma *fall2-intro*[*intro*]:
assumes
 $\bigwedge \pi\ \pi'. \llbracket wfp\ AP'\ \pi; wfp\ AP'\ \pi';$
 $\pi l \neq [] \implies stateOf\ \pi = stateOf\ (last\ \pi l);$
 $\pi l = [] \implies stateOf\ \pi = s0;$

```

      stateOf  $\pi' = \text{stateOf } \pi$ 
    ]
   $\implies F \pi \pi' \pi l$ 
shows fall2  $F \pi l$ 
<proof>

lemma fall2-elim[elim]:
assumes fall2  $F \pi l$ 
and
 $(\bigwedge \pi \pi'. \llbracket \text{wfp } AP' \pi; \text{wfp } AP' \pi';$ 
   $\pi l \neq [] \implies \text{stateOf } \pi = \text{stateOf } (\text{last } \pi l); \pi l = [] \implies \text{stateOf } \pi = s0;$ 
   $\text{stateOf } \pi' = \text{stateOf } \pi$ 
 $\rrbracket$ 
 $\implies F \pi \pi' \pi l)$ 
 $\implies \chi$ 
shows  $\chi$ 
<proof>

end-of-context Shallow

```

4 Noninterference à la Goguen and Meseguer

4.1 Goguen-Meseguer noninterference

Definition

```

locale GM-sec-model =
  fixes st0 :: 'St
  and do :: 'St  $\Rightarrow$  'U  $\Rightarrow$  'C  $\Rightarrow$  'St
  and out :: 'St  $\Rightarrow$  'U  $\Rightarrow$  'Out
  and GH :: 'U set
  and GL :: 'U set
begin

```

Extension of “do” to sequences of pairs (user, command):

```

fun doo :: 'St  $\Rightarrow$  ('U  $\times$  'C) list  $\Rightarrow$  'St where
  doo st [] = st
  |doo st ((u,c) # ucl) = (doo (do st u c) ucl)

```

```

definition purge :: 'U set  $\Rightarrow$  ('U  $\times$  'C) list  $\Rightarrow$  ('U  $\times$  'C) list where
  purge G ucl  $\equiv$  filter ( $\lambda (u,c). u \notin G$ ) ucl

```

```

lemma purge-Nil[simp]: purge G [] = []
and purge-Cons-in[simp]:  $u \notin G \implies \text{purge } G ((u,c) \# ucl) = (u,c) \# \text{purge } G ucl$ 
and purge-Cons-notIn[simp]:  $u \in G \implies \text{purge } G ((u,c) \# ucl) = \text{purge } G ucl$ 
<proof>

```

```

lemma purge-append:
  purge G (ucl1 @ ucl2) = purge G ucl1 @ purge G ucl2

```

$\langle proof \rangle$

definition *nonint* :: *bool* **where**

nonint $\equiv \forall \text{ ucl. } \forall u \in GL. \text{ out } (doo \text{ st0 } ucl) \text{ } u = \text{ out } (doo \text{ st0 } (\text{purge } GH \text{ ucl})) \text{ } u$

end-of-context GM-sec-model

4.2 Specialized Kripke structures

As a preparation for representing noninterference in HyperCTL*, we define a specialized notion of Kripke structure. It is enriched with the following data: two binary state predicates *f* and *g*, intuitively capturing high-input and low-output equivalence, respectively; a set *Sink* of states immediately accessible from any state and such that, for the states in *Sink*, there exist self-transitions and *f* holds.

This specialized structure, represented by the locale *Shallow-Idle*, is an auxiliary that streamlines our proofs, easing the connection between finite paths (specific to Goguen-Meseguer noninterference) and infinite paths (specific to the HyperCTL* semantics). The desired Kripke structure produced from a Goguen-Meseguer model will actually be such a specialized structure.

locale *Shallow-Idle* = *Shallow* *S* *s0* δ *AP*

for *S* :: '*state set* **and** *s0* :: '*state* **and** δ :: '*state* $\Rightarrow$ '*state set*
and *AP* :: '*aprop set*
and *f* :: '*state* $\Rightarrow$ '*state* $\Rightarrow$ *bool* **and** *g* :: '*state* $\Rightarrow$ '*state* $\Rightarrow$ *bool*
and *Sink* :: '*state set*

+

assumes *Sink-S*: *Sink* $\subseteq S$
and *Sink*: $\bigwedge s. s \in S \Longrightarrow \exists s'. s' \in \delta s \cap Sink$
and *Sink-idle*: $\bigwedge s. s \in Sink \Longrightarrow s \in \delta s$
and *Sink-f*: $\bigwedge s1 \ s2. \{s1, s2\} \subseteq Sink \Longrightarrow f \ s1 \ s2$

begin

definition *toSink* *s* $\equiv SOME \ s'. \ s' \in \delta \ s \cap Sink$

lemma *toSink*: $s \in S \Longrightarrow toSink \ s \in \delta \ s \cap Sink$

$\langle proof \rangle$

lemma *fall2-imp-alw*:

fall2 $(\lambda \pi' \pi \pi l. \text{ imp } (alw \ (\varphi \ \pi l)) \ (alw \ (\psi \ \pi l)) \ (\pi l \ @ \ [\pi, \pi'])) \ []$
 $\longleftrightarrow$
 $(\forall \pi \ \pi'. \text{ wfp } AP' \ \pi \wedge \text{ wfp } AP' \ \pi' \wedge \text{ stateOf } \pi = s0 \wedge \text{ stateOf } \pi' = s0$
 $\longrightarrow \text{ imp } (alw \ (\varphi \ [])) \ (alw \ (\psi \ [])) \ [\pi, \pi'])$
 $)$
 $\langle proof \rangle$

lemma *wfp-stateOf-shift-stake-sconst*:

fixes $\pi \ i$
defines $\pi 1 \equiv \text{shift } (\text{stake } (\text{Suc } i) \ \pi) \ (\text{sconst } (\text{toSink } (\text{fst } (\pi !! i))), \ L \ (\text{toSink } (\text{fst } (\pi !! i))))$
assumes π : $\text{wfp } AP' \ \pi$
shows $\text{wfp } AP' \ \pi 1 \wedge \text{stateOf } \pi 1 = \text{stateOf } \pi$
 $\langle \text{proof} \rangle$

lemma *fall2-imp-alm-index*:

assumes 0 : $\bigwedge \pi \ \pi'. \text{wfp } AP' \ \pi \wedge \text{wfp } AP' \ \pi' \longrightarrow$
 $\varphi \ \square \ [\pi, \pi'] = f \ (\text{stateOf } \pi) \ (\text{stateOf } \pi') \wedge$
 $\psi \ \square \ [\pi, \pi'] = g \ (\text{stateOf } \pi) \ (\text{stateOf } \pi')$
shows
 $\text{fall2 } (\lambda \ \pi' \ \pi \ \pi l. \text{imp } (\text{alm } (\varphi \ \pi l)) \ (\text{alm } (\psi \ \pi l)) \ (\pi l \ @ \ [\pi, \pi'])) \ \square$
 $\longleftrightarrow$
 $(\forall \ \pi \ \pi'. \text{wfp } AP' \ \pi \wedge \text{wfp } AP' \ \pi' \wedge \text{stateOf } \pi = s0 \wedge \text{stateOf } \pi' = s0$
 $\longrightarrow$
 $(\forall \ i. (\forall \ j \leq i. f \ (\text{fst } (\pi !! j)) \ (\text{fst } (\pi' !! j))) \longrightarrow g \ (\text{fst } (\pi !! i)) \ (\text{fst } (\pi' !! i)))$
 $)$
(is $?L \longleftrightarrow ?R)$
 $\langle \text{proof} \rangle$

end-of-context Shallow-Idle

4.3 Faithful representation as a HyperCTL* property

Starting with a Goguen-Meseguer model, we will produce a specialized Kripke structure and a shallow HyperCTL* formula. Then we will prove that the structure satisfies the formula iff the Goguen-Meseguer model satisfies noninterference.

The Kripke structure has two kinds of states: “idle” states storing Goguen-Meseguer states, and normal states storing Goguen-Meseguer states, users and commands: the former will be used for synchronization and the latter for Goguen-Meseguer steps. The Kripke labels store user-command actions and user-output observations.

datatype $(\text{'St}, \text{'U}, \text{'C}) \ \text{state} =$
 $\text{isIdle: Idle } (\text{getGMState: 'St}) \mid \text{isState: State } (\text{getGMState: 'St}) \ (\text{getGMUser: 'U}) \ (\text{getGMCom: 'C})$

datatype $(\text{'U}, \text{'C}, \text{'Out}) \ \text{aprop} = \text{Last } \text{'U} \ \text{'C} \mid \text{Obs } \text{'U} \ \text{'Out}$

definition getGMUserCom **where** $\text{getGMUserCom } s = (\text{getGMUser } s, \text{getGMCom } s)$

lemma $\text{getGMUserCom}[\text{simp}]$: $\text{getGMUserCom } (\text{State } st \ u \ c) = (u, c)$
 $\langle \text{proof} \rangle$

context *GM-sec-model*

begin

primrec $L :: (\text{'St}, \text{'U}, \text{'C}) \ \text{state} \Rightarrow (\text{'U}, \text{'C}, \text{'Out}) \ \text{aprop} \ \text{set} \ \text{where}$
 $L \ (\text{Idle } st) = \{\text{Obs } u' \ (\text{out } st \ u') \mid u'. \ \text{True}\}$
 $L \ (\text{State } st \ u \ c) = \{\text{Last } u\} \cup \{\text{Obs } u' \ (\text{out } st \ u') \mid u'. \ \text{True}\}$

Get the Goguen-Meseguer state:

primrec *getGMState* **where**
getGMState (*Idle st*) = *st*
getGMState (*State st u c*) = *st*

lemma *Last-in-L[simp]*: *Last u c ∈ L s* $\longleftrightarrow$ ($\exists$ *st*. *s* = *State st u c*)
 $\langle proof \rangle$

lemma *Obs-in-L[simp]*: *Obs u ou ∈ L s* $\longleftrightarrow$ *ou* = *out (getGMState s) u*
 $\langle proof \rangle$

primrec $\delta :: ('St, 'U, 'C) \text{ state} \Rightarrow ('St, 'U, 'C) \text{ state set}$ **where**
 δ (*Idle st*) = {*Idle st*} $\cup$ {*State (do st u' c')* *u' c' | u' c'. True*}
 δ (*State st u c*) = {*Idle st*} $\cup$ {*State (do st u' c')* *u' c' | u' c'. True*}

abbreviation *s0* **where** *s0* $\equiv$ *State st0 any any*

definition *f* :: (*'a*, *'U*, *'b*) *state* $\Rightarrow$ (*'c*, *'U*, *'b*) *state* $\Rightarrow$ *bool*
where
f s s' $\equiv$
 $\forall u c. u \notin GH \longrightarrow ((\exists st. s = \text{State } st \ u \ c) \longleftrightarrow (\exists st'. s' = \text{State } st' \ u \ c))$

definition *g* :: (*'St*, *'a*, *'b*) *state* $\Rightarrow$ (*'St*, *'c*, *'d*) *state* $\Rightarrow$ *bool*
where
g s s' $\equiv \forall u1. u1 \in GL \longrightarrow \text{out } (\text{getGMState } s) \ u1 = \text{out } (\text{getGMState } s') \ u1$

lemma *f-id[simp,intro!]*: *f s s* $\langle proof \rangle$

definition *Sink* :: (*'St*, *'U*, *'C*) *state set*
where
Sink = {*Idle st | st . True*}

end

sublocale *GM-sec-model* < *Shallow-Idle*
where *S* = *UNIV::('St,'U,'C) state set*
and *AP* = *UNIV::('U,'C,'Out) aprop set* **and** *AP'* = *UNIV::('U,'C,'Out) aprop set*
and *s0* = *s0* **and** *L* = *L* **and** $\delta = \delta$ **and** *f* = *f* **and** *g* = *g* **and** *Sink* = *Sink*
 $\langle proof \rangle$

context *GM-sec-model*
begin

lemma *apropsOf-L-stateOf[simp]*:
wfp AP' $\pi \implies$ apropsOf $\pi = L$ (stateOf π)
 $\langle proof \rangle$

The equality of two states w.r.t. a given “last” user-command pair:

definition $eqOnUC ::$
 $nat \Rightarrow nat \Rightarrow 'U \Rightarrow 'C \Rightarrow (('St, 'U, 'C) \text{ state}, ('U, 'C, 'Out) \text{ apro}) \text{ sfmla}$
where
 $eqOnUC \ i \ i' \ u \ c \equiv eq \ (atom \ (Last \ u \ c) \ i) \ (atom \ (Last \ u \ c) \ i')$

The equality of two states w.r.t. all their “last” user-command pairs with the user not in GH:

definition $eqButGH ::$
 $nat \Rightarrow nat \Rightarrow (('St, 'U, 'C) \text{ state}, ('U, 'C, 'Out) \text{ apro}) \text{ sfmla}$
where
 $eqButGH \ i \ i' \equiv scon \ \{eqOnUC \ i \ i' \ u \ c \mid u \ c. \ (u, c) \in (UNIV - GH) \times UNIV\}$

The equality of two states w.r.t. a given “observed” user-observation pair:

definition $eqOnUOut ::$
 $nat \Rightarrow nat \Rightarrow 'U \Rightarrow 'Out \Rightarrow (('St, 'U, 'C) \text{ state}, ('U, 'C, 'Out) \text{ apro}) \text{ sfmla}$
where
 $eqOnUOut \ i \ i' \ u \ ou \equiv eq \ (atom \ (Obs \ u \ ou) \ i) \ (atom \ (Obs \ u \ ou) \ i')$

The equality of two states w.r.t. all their “observed” user-observation pairs with the user in GL:

definition $eqOnGL ::$
 $nat \Rightarrow nat \Rightarrow (('St, 'U, 'C) \text{ state}, ('U, 'C, 'Out) \text{ apro}) \text{ sfmla}$
where
 $eqOnGL \ i \ i' \equiv scon \ \{eqOnUOut \ i \ i' \ u \ ou \mid u \ ou. \ (u, ou) \in GL \times UNIV\}$

lemma $eqOnUC-0-Suc0[simp]$:
assumes $wfp \ AP' \ \pi$ **and** $wfp \ AP' \ \pi'$
shows
 $eqOnUC \ 0 \ (Suc \ 0) \ u \ c \ [\pi, \pi']$
 $\longleftrightarrow$
 $((\exists \ st. \ stateOf \ \pi = State \ st \ u \ c) =$
 $(\exists \ st'. \ stateOf \ \pi' = State \ st' \ u \ c)$
 $)$
 $\langle proof \rangle$

lemma $eqOnUOut-0-Suc0[simp]$:
assumes $wfp \ AP' \ \pi$ **and** $wfp \ AP' \ \pi'$
shows
 $eqOnUOut \ 0 \ (Suc \ 0) \ u \ ou \ [\pi, \pi']$
 $\longleftrightarrow$
 $(ou = out \ (getGMState \ (stateOf \ \pi)) \ u \longleftrightarrow$
 $ou = out \ (getGMState \ (stateOf \ \pi')) \ u$
 $)$
 $\langle proof \rangle$

The (shallow) noninterference formula – it will be proved equivalent to nonint, the original statement of noninterference.

definition $nonintSfmla :: (('St, 'U, 'C) \text{ state}, ('U, 'C, 'Out) \text{ apro}) \text{ sfmla}$ **where**
 $nonintSfmla \equiv$
 $fall2 \ (\lambda \ \pi' \ \pi \ \pi l.$

```

    imp (alw (eqButGH (length  $\pi l$ ) (Suc (length  $\pi l$ ))))
      (alw (eqOnGL (length  $\pi l$ ) (Suc (length  $\pi l$ ))))
    ( $\pi l$  @ [ $\pi, \pi$ ])
  )

```

First, we show that `nonintSfmla` is equivalent to `nonintSI`, a variant of noninterference that speaks about Synchronized Infinite paths.

definition `nonintSI :: bool` **where**

```

nonintSI  $\equiv$ 
 $\forall \pi \pi'. \text{wfp UNIV } \pi \wedge \text{wfp UNIV } \pi' \wedge \text{stateOf } \pi = s0 \wedge \text{stateOf } \pi' = s0$ 
 $\longrightarrow$ 
 $(\forall i. (\forall j \leq i. f (\text{fst } (\pi !! j)) (\text{fst } (\pi' !! j))) \longrightarrow g (\text{fst } (\pi !! i)) (\text{fst } (\pi' !! i)))$ 

```

lemma `nonintSfmla-nonintSI: nonintSfmla []  $\longleftrightarrow$  nonintSI`
`<proof>`

In turn, `nonintSI` will be shown equivalent to `nonintS`, a variant speaking about Synchronized finite paths. To this end, we introduce a notion of well-formed finite path (`wffp`) – besides finiteness, another difference from the previously defined infinite paths is that, thanks to the fact that here AP coincides with AP', paths are mere sequences of states as opposed to pairs (state, set of atomic predicates).

inductive `wffp :: ('St, 'U, 'C) state list  $\Rightarrow$  bool`

where

```

Singl[simp,intro!]: wffp [s]
|
Cons[intro]:
 $\llbracket s' \in \delta \ s; \text{wffp } (s' \# \text{sl}) \rrbracket$ 
 $\implies$ 
 $\text{wffp } (s \# s' \# \text{sl})$ 

```

lemma `wffp-induct2[consumes 1, case-names Singl Cons]:`

assumes `wffp sl`

and $\bigwedge s. P [s]$

and $\bigwedge s \text{sl}. \llbracket \text{hd } \text{sl} \in \delta \ s; \text{wffp } \text{sl}; P \ \text{sl} \rrbracket \implies P (s \# \text{sl})$

shows `P sl`

`<proof>`

definition `nonintS :: bool` **where**

```

nonintS  $\equiv$ 
 $\forall \text{sl sl}'. \text{wffp } \text{sl} \wedge \text{wffp } \text{sl}' \wedge \text{hd } \text{sl} = s0 \wedge \text{hd } \text{sl}' = s0 \wedge$ 
 $\text{list-all2 } f \ \text{sl } \text{sl}' \longrightarrow g (\text{last } \text{sl}) (\text{last } \text{sl}')$ 

```

lemma `wffp-NE: assumes wffp sl shows sl  $\neq$  []`

`<proof>`

lemma `wffp:`

```

wffp sl  $\longleftrightarrow$   $\text{sl} \neq [] \wedge (\forall i. \text{Suc } i < \text{length } \text{sl} \longrightarrow \text{sl}!(\text{Suc } i) \in \delta(\text{sl}!i))$ 
(is ?L  $\longleftrightarrow$  ?A  $\wedge (\forall i. ?R \ i)$ )

```

$\langle proof \rangle$

lemma *wffp-hdI*[*intro*]:
assumes *wffp sl* **and** *hd sl* $\in \delta$ *s*
shows *wffp (s # sl)*
 $\langle proof \rangle$

lemma *wffp-append*:
assumes *sl: wffp sl* **and** *sl1: wffp sl1* **and** *h: hd sl1* $\in \delta$ (*last sl*)
shows *wffp (sl @ sl1)*
 $\langle proof \rangle$

lemma *wffp-append-iff*:
wffp (sl @ sl1) $\longleftrightarrow$
(*wffp sl* $\wedge$ *sl1* = []) $\vee$
(*sl* = [] $\wedge$ *wffp sl1*) $\vee$
(*wffp sl* $\wedge$ *wffp sl1* $\wedge$ *hd sl1* $\in \delta$ (*last sl*))
(**is** - $\longleftrightarrow$?*R*)
 $\langle proof \rangle$

lemma *wffp-to-wfp*:
assumes π -def: $\pi = \text{map } (\lambda s. (s, L s)) \text{ sl } @ - \text{ sconst } (toSink (last sl), L (toSink (last sl)))$
assumes *sl: wffp sl*
shows
wfp UNIV π $\wedge$
($\forall i < \text{length } sl. sl ! i = \text{fst } (\pi !! i)$) $\wedge$
($\forall i \geq \text{length } sl. \text{fst } (\pi !! i) = \text{toSink } (last sl)$) $\wedge$
stateOf $\pi = \text{hd } sl$
 $\langle proof \rangle$

lemma *wffp-imp-appendL*: *wffp (sl1 @ sl2)* $\implies sl1 \neq [] \implies \text{wffp } sl1$
 $\langle proof \rangle$

lemma *wffp-imp-appendR*: *wffp (sl1 @ sl2)* $\implies sl2 \neq [] \implies \text{wffp } sl2$
 $\langle proof \rangle$

lemma *wffp-iff-map-Idle*:
assumes *wffp sl*
shows
 $\exists n \text{ st.}$
($n > 0 \wedge sl = \text{map Idle } (\text{replicate } n \text{ st})$) $\vee$
($\exists st1 \text{ u1 c1 } sl1. sl = \text{map Idle } (\text{replicate } n \text{ st}) @ [\text{State } st1 \text{ u1 c1}] @ sl1$)
 $\langle proof \rangle$

lemma *wffp-cases3*[*elim, consumes 1, case-names Idle State Idle-State*]:
assumes *wffp sl*
obtains
n st **where**
 $n > 0$ **and** $sl = \text{map Idle } (\text{replicate } n \text{ st})$

|
st u c sl1 **where**
sl = State st u c # sl1 **and** *sl1 ≠ []* $\implies$ *wffp sl1 ∧ hd sl1 ∈ δ (State st u c)*
 |
n st u c sl1 **where**
n > 0 **and** *sl = map Idle (replicate n st) @ [State (do st u c) u c] @ sl1*
and *sl1 ≠ []* $\implies$ *wffp sl1 ∧ hd sl1 ∈ δ (State (do st u c) u c)*
 ⟨*proof*⟩

lemma *wffp-cases2[elim, consumes 1, case-names Idle State]*:
assumes *wffp sl*
obtains
n st **where**
n > 0 **and** *sl = map Idle (replicate n st)*
 |
n st st1 u c sl1 **where**
sl = map Idle (replicate n st) @ [State st1 u c] @ sl1
and *sl1 ≠ []* $\implies$ *wffp sl1 ∧ hd sl1 ∈ δ (State st1 u c)*
 ⟨*proof*⟩

lemma *wffp-Idle-Idle*:
assumes *wffp (sl1 @ [Idle st1] @ [Idle st2] @ sl2)*
shows *st2 = st1*
 ⟨*proof*⟩

lemma *wffp-Idle-State*:
assumes *wffp (sl1 @ [Idle st1] @ [State st2 u2 c2] @ sl2)*
shows *st2 = st1 ∨ st2 = do st1 u2 c2*
 ⟨*proof*⟩

lemma *wffp-State-Idle*:
assumes *wffp (sl1 @ [State st1 u1 c1] @ [Idle st2] @ sl2)*
shows *st2 = st1*
 ⟨*proof*⟩

lemma *wffp-State-State*:
assumes *wffp (sl1 @ [State st1 u1 c1] @ [State st2 u2 c2] @ sl2)*
shows *st2 = do st1 u2 c2*
 ⟨*proof*⟩

lemma *wfp-to-wffp*:
assumes *sl-def: sl = map fst (stake i π)* **and** *i: i > 0* **and** *π: wfp UNIV π*
shows
wffp sl ∧
(∀ j < length sl. fst (π !! j) = sl ! j) ∧
stateOf π = hd sl
 ⟨*proof*⟩

lemma *nonintSI-nonintS*: *nonintSI* $\longleftrightarrow$ *nonintS*

$\langle proof \rangle$

Finally, we show that `nonintS` is equivalent to standard noninterference (predicate `nonint`).

`purgeIdle` removes the idle steps from a finite path:

definition $purgeIdle :: ('St, 'U, 'C) \text{ state list} \Rightarrow ('St, 'U, 'C) \text{ state list}$
where $purgeIdle \equiv filter \ isState$

lemma $purgeIdle-simps[simp]$:
 $purgeIdle [] = []$
 $purgeIdle ((Idle\ st) \# sl) = purgeIdle\ sl$
 $purgeIdle ((State\ st\ u\ c) \# sl) = (State\ st\ u\ c) \# purgeIdle\ sl$
 $\langle proof \rangle$

lemma $purgeIdle-append$:
 $purgeIdle (sl1 @ sl2) = purgeIdle\ sl1 @ purgeIdle\ sl2$
 $\langle proof \rangle$

lemma $purgeIdle-set-isState$:
assumes $s \in set\ (purgeIdle\ sl)$
shows $isState\ s$
 $\langle proof \rangle$

lemma $purgeIdle-Nil-iff$:
 $purgeIdle\ sl = [] \iff (\forall s \in set\ sl. \neg isState\ s)$
 $\langle proof \rangle$

lemma $purgeIdle-Cons-iff$:
 $purgeIdle\ sl = s \# sl1$
 $\iff$
 $(\exists sl1\ sl2. sl = sl1 @ s \# sl2 \wedge$
 $(\forall s1 \in set\ sl1. \neg isState\ s1) \wedge isState\ s \wedge purgeIdle\ sl2 = sl1)$
 $\langle proof \rangle$

lemma $purgeIdle-map-Idle[simp]$:
 $purgeIdle (map\ Idle\ s) = []$
 $\langle proof \rangle$

lemma $purgeIdle-replicate-Idle[simp]$:
 $purgeIdle (replicate\ n\ (Idle\ st)) = []$
 $\langle proof \rangle$

lemma $wffp-purgeIdle-Nil$:
assumes $wffp\ sl$ **and** $purgeIdle\ sl = []$
shows $\exists n\ st. n > 0 \wedge sl = replicate\ n\ (Idle\ st)$
 $\langle proof \rangle$

lemma $wffp-hd-purgeIdle$:
assumes $wsl: wffp\ sl$ **and** $psl: purgeIdle\ sl \neq []$
and $ist: isState\ s$ **and** $hsl: hd\ sl \in \delta\ s$

shows $hd\ (purgeIdle\ sl) \in \delta\ s$
 $\langle proof \rangle$

lemma *wffp-purgeIdle*:
assumes $wffp\ sl$ **and** $purgeIdle\ sl \neq []$
shows $wffp\ (purgeIdle\ sl)$
 $\langle proof \rangle$

lemma *isState-purgeIdle*:
 $(\exists\ sl.\ purgeIdle\ sl = sll) \longleftrightarrow list-all\ isState\ sll$
 $\langle proof \rangle$

lemma *wffp-last-purgeIdle*:
assumes $wffp\ sl$ **and** $purgeIdle\ sl \neq []$
shows $getGMState\ (last\ (purgeIdle\ sl)) = getGMState\ (last\ sl)$
 $\langle proof \rangle$

lemma *wffp-isState-doo*:
assumes $wffp\ sl$ **and** $list-all\ isState\ sl$
shows $doo\ (getGMState\ (hd\ sl))\ (map\ getGMUserCom\ (tl\ sl)) = getGMState\ (last\ sl)$
 $\langle proof \rangle$

lemma *isState-hd-purgeIdle*:
assumes $wsl: wffp\ sl$ **and** $ist: isState\ (hd\ sl)$
shows $purgeIdle\ sl \neq [] \wedge hd\ (purgeIdle\ sl) = hd\ sl$
 $\langle proof \rangle$

lemma *wffp-isState-doo-purgeIdle*:
fixes sl **defines** $sll: sll \equiv purgeIdle\ sl$
assumes $wsl: wffp\ sl$ **and** $ist: isState\ (hd\ sl)$
shows $doo\ (getGMState\ (hd\ sl))\ (map\ getGMUserCom\ (tl\ sll)) = getGMState\ (last\ sl)$
 $\langle proof \rangle$

lemma *map-getGMUserCom-surj*:
assumes $isState\ s$
shows $\exists\ sl.\ wffp\ sl \wedge list-all\ isState\ sl \wedge hd\ sl = s \wedge map\ getGMUserCom\ (tl\ sl) = ucl$
 $\langle proof \rangle$

lemma *purgeIdle-purge-ex*:
assumes $wffp\ sl$ **and** $list-all\ isState\ sl$ **and** $map\ getGMUserCom\ (tl\ sl) = ucl$
shows $\exists\ sl'.\ hd\ sl' = ss' \wedge wffp\ sl' \wedge$
 $list-all2\ f\ (tl\ sl)\ (tl\ sl') \wedge$
 $map\ getGMUserCom\ (purgeIdle\ (tl\ sl')) = purge\ GH\ ucl$
 $\langle proof \rangle$

lemma *purgeIdle-getGMUserCom-purge*:
fixes $sl\ sl'$
defines $ucl \equiv map\ getGMUserCom\ (purgeIdle\ (tl\ sl))$
and $ucl' \equiv map\ getGMUserCom\ (purgeIdle\ (tl\ sl'))$

assumes $wsl: wffp\ sl$ **and** $wsl': wffp\ sl'$ **and** $f: list-all2\ f\ sl\ sl'$
shows $purge\ GH\ ucl = purge\ GH\ ucl'$
 $\langle proof \rangle$

lemma $nonintS\text{-}iff\text{-}nonint$:
 $nonintS \longleftrightarrow nonint$
 $\langle proof \rangle$

theorem $nonintSfmla\text{-}iff\text{-}nonint$:
 $nonintSfmla\ [] \longleftrightarrow nonint$
 $\langle proof \rangle$

end-of-context GM-sec-model

5 Deep representation of HyperCTL* – syntax and semantics

5.1 Path variables and environments

datatype $pvar = Pvariable\ (natOf : nat)$

Deeply embedded (syntactic) formulas

datatype $'aprop\ dfmla =$
 $Atom\ 'aprop\ pvar \mid$
 $Fls \mid Neg\ 'aprop\ dfmla \mid Dis\ 'aprop\ dfmla\ 'aprop\ dfmla \mid$
 $Next\ 'aprop\ dfmla \mid Until\ 'aprop\ dfmla\ 'aprop\ dfmla \mid$
 $Exi\ pvar\ 'aprop\ dfmla$

Derived operators

definition $Tr \equiv Neg\ Fls$
definition $Con\ \varphi\ \psi \equiv Neg\ (Dis\ (Neg\ \varphi)\ (Neg\ \psi))$
definition $Imp\ \varphi\ \psi \equiv Dis\ (Neg\ \varphi)\ \psi$
definition $Eq\ \varphi\ \psi \equiv Con\ (Imp\ \varphi\ \psi)\ (Imp\ \psi\ \varphi)$
definition $Fall\ p\ \varphi \equiv Neg\ (Exi\ p\ (Neg\ \varphi))$
definition $Ev\ \varphi \equiv Until\ Tr\ \varphi$
definition $Alw\ \varphi \equiv Neg\ (Ev\ (Neg\ \varphi))$
definition $Wuntil\ \varphi\ \psi \equiv Dis\ (Until\ \varphi\ \psi)\ (Alw\ \varphi)$
definition $Fall2\ p\ p'\ \varphi \equiv Fall\ p\ (Fall\ p'\ \varphi)$

lemmas $der\text{-}Op\text{-}defs =$
 $Tr\text{-}def\ Con\text{-}def\ Imp\text{-}def\ Eq\text{-}def\ Ev\text{-}def\ Alw\text{-}def\ Wuntil\text{-}def\ Fall\text{-}def\ Fall2\text{-}def$

Well-formed formulas are those that do not have a temporal operator outside the scope of any quantifier – indeed, in HyperCTL* such a situation does not make sense, since the temporal operators refer to previously introduced/quantified paths.

primrec $wff :: 'aprop\ dfmla \Rightarrow bool$ **where**
 $wff\ (Atom\ a\ p) = True$
 $wff\ Fls = True$
 $wff\ (Neg\ \varphi) = wff\ \varphi$

$|wff\ (Dis\ \varphi\ \psi) = (wff\ \varphi \wedge wff\ \psi)$
 $|wff\ (Next\ \varphi) = False$
 $|wff\ (Until\ \varphi\ \psi) = False$
 $|wff\ (Exi\ p\ \varphi) = True$

The ability to pick a fresh variable

lemma *finite-fresh-pvar*:
assumes *finite* ($P :: pvar\ set$)
obtains p **where** $p \notin P$
 $\langle proof \rangle$

definition *getFresh* $:: pvar\ set \Rightarrow pvar$ **where**
 $getFresh\ P \equiv SOME\ p.\ p \notin P$

lemma *getFresh*:
assumes *finite* P **shows** $getFresh\ P \notin P$
 $\langle proof \rangle$

The free-variables operator

primrec *FV* $:: 'a\ prop\ dfmla \Rightarrow pvar\ set$ **where**
 $FV\ (Atom\ a\ p) = \{p\}$
 $|FV\ Fls = \{\}$
 $|FV\ (Neg\ \varphi) = FV\ \varphi$
 $|FV\ (Dis\ \varphi\ \psi) = FV\ \varphi \cup FV\ \psi$
 $|FV\ (Next\ \varphi) = FV\ \varphi$
 $|FV\ (Until\ \varphi\ \psi) = FV\ \varphi \cup FV\ \psi$
 $|FV\ (Exi\ p\ \varphi) = FV\ \varphi - \{p\}$

Environments

type-synonym *env* $= pvar \Rightarrow nat$

definition *eqOn* $P\ env\ env1 \equiv \forall\ p.\ p \in P \longrightarrow env\ p = env1\ p$

lemma *eqOn-Un[simp]*:
 $eqOn\ (P \cup Q)\ env\ env1 \longleftrightarrow eqOn\ P\ env\ env1 \wedge eqOn\ Q\ env\ env1$
 $\langle proof \rangle$

lemma *eqOn-update[simp]*:
 $eqOn\ P\ (env(p := \pi))\ (env1(p := \pi)) \longleftrightarrow eqOn\ (P - \{p\})\ env\ env1$
 $\langle proof \rangle$

lemma *eqOn-singl[simp]*:
 $eqOn\ \{p\}\ env\ env1 \longleftrightarrow env\ p = env1\ p$
 $\langle proof \rangle$

context *Shallow*
begin

5.2 The semantic operator

The semantics will interpret deep (syntactic) formulas as shallow formulas. Recall that the latter are predicates on lists of paths – the interpretation will be parameterized by an environment mapping each path variable to a number indicating the index (in the list) for the path denoted by the variable.

The semantics will only be meaningful if the indexes of a formula’s free variables are smaller than the length of the path list – we call this property “compatibility”.

primrec *sem* :: 'aprop dfmla $\Rightarrow$ env $\Rightarrow$ ('state,'aprop) sfmla **where**
sem (Atom *a p*) env = atom *a* (env *p*)
sem Fls env = fls
sem (Neg φ) env = neg (*sem* φ env)
sem (Dis $\varphi \psi$) env = dis (*sem* φ env) (*sem* ψ env)
sem (Next φ) env = next (*sem* φ env)
sem (Until $\varphi \psi$) env = until (*sem* φ env) (*sem* ψ env)
sem (Exi $p \varphi$) env = exi ($\lambda \pi \pi l. \text{sem } \varphi (\text{env}(p := \text{length } \pi l)) (\pi l @ [\pi])$)

lemma *sem-Exi-explicit*:

sem (Exi $p \varphi$) env $\pi l \longleftrightarrow$
 $(\exists \pi. \text{wfp } AP' \pi \wedge \text{stateOf } \pi = (\text{if } \pi l \neq [] \text{ then stateOf } (\text{last } \pi l) \text{ else } s0) \wedge$
 $\text{sem } \varphi (\text{env}(p := \text{length } \pi l)) (\pi l @ [\pi]))$
 <proof>

lemma *sem-derived[simp]*:

sem Tr env = tr
sem (Con $\varphi \psi$) env = con (*sem* φ env) (*sem* ψ env)
sem (Imp $\varphi \psi$) env = imp (*sem* φ env) (*sem* ψ env)
sem (Eq $\varphi \psi$) env = eq (*sem* φ env) (*sem* ψ env)
sem (Fall $p \varphi$) env = fall ($\lambda \pi \pi l. \text{sem } \varphi (\text{env}(p := \text{length } \pi l)) (\pi l @ [\pi])$)
sem (Ev φ) env = ev (*sem* φ env)
sem (Alw φ) env = alw (*sem* φ env)
sem (Wuntil $\varphi \psi$) env = wuntil (*sem* φ env) (*sem* ψ env)
 <proof>

lemma *sem-Fall2[simp]*:

sem (Fall2 $p p' \varphi$) env =
 fall2 ($\lambda \pi' \pi \pi l. \text{sem } \varphi (\text{env } (p := \text{length } \pi l, p' := \text{Suc}(\text{length } \pi l))) (\pi l @ [\pi, \pi'])$)
 <proof>

Compatibility of a pair (environment, path list) on a set of variables means no out-of-range references:

definition *cpt* *P* env $\pi l \equiv \forall p \in P. \text{env } p < \text{length } \pi l$

lemma *cpt-Un[simp]*:

cpt (*P* $\cup$ *Q*) env $\pi l \longleftrightarrow \text{cpt } P \text{ env } \pi l \wedge \text{cpt } Q \text{ env } \pi l$
 <proof>

lemma *cpt-singl[simp]*:

cpt {*p*} env $\pi l \longleftrightarrow \text{env } p < \text{length } \pi l$
 <proof>

lemma *cpt-map-stl[simp]*:
 $\text{cpt } P \text{ env } \pi l \implies \text{cpt } P \text{ env } (\text{map stl } \pi l)$
 $\langle \text{proof} \rangle$

Next we prove that the semantics of well-formed formulas only depends on the interpretation of the free variables of a formula and on the current state of the last recorded path – we call the latter the “pointer” of the path list.

fun *pointerOf* :: ('state,'aprop) path list $\Rightarrow$ 'state **where**
pointerOf $\pi l = (\text{if } \pi l \neq [] \text{ then stateOf (last } \pi l) \text{ else } s0)$

Equality of two pairs (environment,path list) on a set of variables:

definition *eqOn* ::
 $pvar \text{ set} \Rightarrow env \Rightarrow ('state,'aprop) \text{ path list} \Rightarrow env \Rightarrow ('state,'aprop) \text{ path list} \Rightarrow bool$
where
 $\text{eqOn } P \text{ env } \pi l \text{ env1 } \pi l1 \equiv \forall p. p \in P \longrightarrow \pi l!(\text{env } p) = \pi l1!(\text{env1 } p)$

lemma *eqOn-Un[simp]*:
 $\text{eqOn } (P \cup Q) \text{ env } \pi l \text{ env1 } \pi l1 \longleftrightarrow \text{eqOn } P \text{ env } \pi l \text{ env1 } \pi l1 \wedge \text{eqOn } Q \text{ env } \pi l \text{ env1 } \pi l1$
 $\langle \text{proof} \rangle$

lemma *eqOn-singl[simp]*:
 $\text{eqOn } \{p\} \text{ env } \pi l \text{ env1 } \pi l1 \longleftrightarrow \pi l!(\text{env } p) = \pi l1!(\text{env1 } p)$
 $\langle \text{proof} \rangle$

lemma *eqOn-map-stl[simp]*:
 $\text{cpt } P \text{ env } \pi l \implies \text{cpt } P \text{ env1 } \pi l1 \implies$
 $\text{eqOn } P \text{ env } \pi l \text{ env1 } \pi l1 \implies \text{eqOn } P \text{ env } (\text{map stl } \pi l) \text{ env1 } (\text{map stl } \pi l1)$
 $\langle \text{proof} \rangle$

lemma *cpt-map-sdrop[simp]*:
 $\text{cpt } P \text{ env } \pi l \implies \text{cpt } P \text{ env } (\text{map (sdrop } i) \pi l)$
 $\langle \text{proof} \rangle$

lemma *cpt-update[simp]*:
assumes $\text{cpt } (P - \{p\}) \text{ env } \pi l$
shows $\text{cpt } P (\text{env}(p := \text{length } \pi l)) (\pi l @ [\pi])$
 $\langle \text{proof} \rangle$

lemma *eqOn-map-sdrop[simp]*:
 $\text{cpt } V \text{ env } \pi l \implies \text{cpt } V \text{ env1 } \pi l1 \implies$
 $\text{eqOn } V \text{ env } \pi l \text{ env1 } \pi l1 \implies \text{eqOn } V \text{ env } (\text{map (sdrop } i) \pi l) \text{ env1 } (\text{map (sdrop } i) \pi l1)$
 $\langle \text{proof} \rangle$

lemma *eqOn-update[simp]*:
assumes $\text{cpt } (P - \{p\}) \text{ env } \pi l$ **and** $\text{cpt } (P - \{p\}) \text{ env1 } \pi l1$
shows
 $\text{eqOn } P (\text{env}(p := \text{length } \pi l)) (\pi l @ [\pi]) (\text{env1}(p := \text{length } \pi l1)) (\pi l1 @ [\pi])$
 $\longleftrightarrow$

$eqOn (P - \{p\}) \text{ env } \pi l \text{ env1 } \pi l1$
 $\langle proof \rangle$

lemma *eqOn-FV-sem-NE*:

assumes $cpt (FV \varphi) \text{ env } \pi l$ **and** $cpt (FV \varphi) \text{ env1 } \pi l1$ **and** $eqOn (FV \varphi) \text{ env } \pi l \text{ env1 } \pi l1$
and $\pi l \neq []$ **and** $\pi l1 \neq []$ **and** $last \pi l = last \pi l1$
shows $sem \varphi \text{ env } \pi l = sem \varphi \text{ env1 } \pi l1$
 $\langle proof \rangle$

The next theorem states that the semantics of a formula on an environment and a list of paths only depends on the pointer of the list of paths.

theorem *eqOn-FV-sem*:

assumes $wff \varphi$ **and** $pointerOf \pi l = pointerOf \pi l1$
and $cpt (FV \varphi) \text{ env } \pi l$ **and** $cpt (FV \varphi) \text{ env1 } \pi l1$ **and** $eqOn (FV \varphi) \text{ env } \pi l \text{ env1 } \pi l1$
shows $sem \varphi \text{ env } \pi l = sem \varphi \text{ env1 } \pi l1$
 $\langle proof \rangle$

corollary *FV-sem*:

assumes $wff \varphi$ **and** $\forall p \in FV \varphi. \text{ env } p = \text{ env1 } p$
and $cpt (FV \varphi) \text{ env } \pi l$ **and** $cpt (FV \varphi) \text{ env1 } \pi l$
shows $sem \varphi \text{ env } \pi l = sem \varphi \text{ env1 } \pi l$
 $\langle proof \rangle$

As a consequence, the interpretation of a closed formula (i.e., a formula with no free variables) will not depend on the environment and, from the list of paths, will only depend on its pointer:

corollary *interp-closed*:

assumes $wff \varphi$ **and** $FV \varphi = \{\}$ **and** $pointerOf \pi l = pointerOf \pi l1$
shows $sem \varphi \text{ env } \pi l = sem \varphi \text{ env1 } \pi l1$
 $\langle proof \rangle$

Therefore, it makes sense to define the interpretation of a closed formula by choosing any environment and any list of paths such that its pointer is the initial state (e.g., the empty list) – knowing that the choices are irrelevant.

definition $semClosed \varphi \equiv sem \varphi (any::env) (SOME \pi l. pointerOf \pi l = s0)$

lemma *semClosed*:

assumes $\varphi: wff \varphi$ $FV \varphi = \{\}$ **and** $p: pointerOf \pi l = s0$
shows $semClosed \varphi = sem \varphi \text{ env } \pi l$
 $\langle proof \rangle$

lemma *semClosed-Nil*:

assumes $\varphi: wff \varphi$ $FV \varphi = \{\}$
shows $semClosed \varphi = sem \varphi \text{ env } []$
 $\langle proof \rangle$

5.3 The conjunction of a finite set of formulas

This is defined by making the set into a list (by choosing any ordering of the elements) and iterating binary conjunction.

definition $Scon :: 'aprop\ dfmla\ set \Rightarrow 'aprop\ dfmla$ **where**
 $Scon\ \varphi s \equiv foldr\ Con\ (asList\ \varphi s)\ Tr$

lemma $sem\text{-}Scon[simp]$:
assumes $finite\ \varphi s$
shows $sem\ (Scon\ \varphi s)\ env = scon\ ((\lambda\ \varphi.\ sem\ \varphi\ env)\ \varphi s)$
 $\langle proof \rangle$

lemma $FV\text{-}Scon[simp]$:
assumes $finite\ \varphi s$
shows $FV\ (Scon\ \varphi s) = \bigcup\ (FV\ \varphi\ \varphi s)$
 $\langle proof \rangle$

end-of-context Shallow

6 Noninterference for models with finitely many users, commands and outputs

In the Noninterference section, we showed how to express Goguen-Meseguer noninterference as a shallow HyperCTL* formula. Here we show that, if one assumes finiteness of the sets of users, commands and outputs, then one can express the property as (the denotation of) a syntactic formula. Note that we do *not* need to assume the state space finite – this is important for a potential application to infinite-state systems.

The Goguen-Meseguer security model with finiteness assumptions

locale $GM\text{-}sec\text{-}model\text{-}finite = GM\text{-}sec\text{-}model\ st0\ do\ out$
for $st0 :: 'St$
and $do :: 'St \Rightarrow 'U \Rightarrow 'C \Rightarrow 'St$
and $out :: 'St \Rightarrow 'U \Rightarrow 'Out$
 $+$
assumes $finite\text{-}U$: $finite\ (UNIV :: 'U\ set)$
and $finite\text{-}C$: $finite\ (UNIV :: 'C\ set)$
and $finite\text{-}Out$: $finite\ (UNIV :: 'Out\ set)$
begin

lemma $finite\text{-}UminusGH$: $finite\ (UNIV - GH)$
 $\langle proof \rangle$

lemma $finite\text{-}GL$: $finite\ GL$
 $\langle proof \rangle$

definition $EqOnUC ::$
 $pvar \Rightarrow pvar \Rightarrow 'U \Rightarrow 'C \Rightarrow ('U, 'C, 'Out)\ apropos\ dfmla$

where

$EqOnUC\ p\ p'\ u\ c \equiv Eq\ (Atom\ (Last\ u\ c)\ p)\ (Atom\ (Last\ u\ c)\ p')$

lemma $EqOnUC$ - $eqOnUC$ [simp]:

assumes $env\ p = i$ **and** $env\ p' = i'$

shows $sem\ (EqOnUC\ p\ p'\ u\ c)\ env = eqOnUC\ i\ i'\ u\ c$

$\langle proof \rangle$

definition $EqButGH$::

$pvar \Rightarrow pvar \Rightarrow ('U, 'C, 'Out)\ aprop\ dfmla$

where

$EqButGH\ p\ p' \equiv Scon\ \{EqOnUC\ p\ p'\ u\ c \mid u\ c.\ (u, c) \in (UNIV - GH) \times UNIV\}$

lemma $finite$ - $EqButGH$:

$finite\ \{EqOnUC\ p\ p'\ u\ c \mid u\ c.\ (u, c) \in (UNIV - GH) \times UNIV\}$ **(is finite ?K)**

$\langle proof \rangle$

lemma $EqButGH$ - $eqButGH$ [simp]:

assumes $env\ p = i$ **and** $env\ p' = i'$

shows $sem\ (EqButGH\ p\ p')\ env = eqButGH\ i\ i'$

$\langle proof \rangle$

lemma FV - $EqButGH$: $FV\ (EqButGH\ p\ p') \subseteq \{p, p'\}$ **(is ?L $\subseteq$?R)**

$\langle proof \rangle$

definition $EqOnUOut$::

$pvar \Rightarrow pvar \Rightarrow 'U \Rightarrow 'Out \Rightarrow ('U, 'C, 'Out)\ aprop\ dfmla$

where

$EqOnUOut\ p\ p'\ u\ ou \equiv Eq\ (Atom\ (Obs\ u\ ou)\ p)\ (Atom\ (Obs\ u\ ou)\ p')$

lemma $EqOnUOut$ - $eqOnUOut$ [simp]:

assumes $env\ p = i$ **and** $env\ p' = i'$

shows $sem\ (EqOnUOut\ p\ p'\ u\ ou)\ env = eqOnUOut\ i\ i'\ u\ ou$

$\langle proof \rangle$

definition $EqOnGL$::

$pvar \Rightarrow pvar \Rightarrow ('U, 'C, 'Out)\ aprop\ dfmla$

where

$EqOnGL\ p\ p' \equiv Scon\ \{EqOnUOut\ p\ p'\ u\ ou \mid u\ ou.\ (u, ou) \in GL \times UNIV\}$

lemma $finite$ - $EqOnGL$:

$finite\ \{EqOnUOut\ p\ p'\ u\ ou \mid u\ ou.\ (u, ou) \in GL \times UNIV\}$ **(is finite ?K)**

$\langle proof \rangle$

lemma $EqOnGL$ - $eqOnGL$ [simp]:

assumes $env\ p = i$ **and** $env\ p' = i'$

shows $sem\ (EqOnGL\ p\ p')\ env = eqOnGL\ i\ i'$

$\langle proof \rangle$

lemma *FV-EqOnGL*: $FV (EqOnGL\ p\ p') \subseteq \{p, p'\}$ (is ?L $\subseteq$?R)
 $\langle proof \rangle$

definition $p0 = getFresh\ \{\}$
definition $p1 = getFresh\ \{p0\}$

lemma $p0p1[simp]$: $p0 \neq p1$ $\langle proof \rangle$

definition $nonintDfmla :: ('U, 'C, 'Out) \rightarrow dfmla$ **where**
 $nonintDfmla \equiv$
 $Fall2\ p0\ p1\ (Imp\ (Alw\ (EqButGH\ p0\ p1))\ (Alw\ (EqOnGL\ p0\ p1)))$

lemma *sem-nonintDfmla*: $sem\ nonintDfmla\ env = nonintSfmla$
 $\langle proof \rangle$

lemma *woff-nonintDfmla[simp]*: $woff\ nonintDfmla$
 $\langle proof \rangle$

lemma *closed-nonintDfmla[simp]*: $FV\ nonintDfmla = \{\}$
 $\langle proof \rangle$

In the end, we obtain that the semantics of the closed (syntactic) formula $nonintDfmla$ expresses noninterference faithfully:

theorem *semClosed-nonintDfmla*: $semClosed\ nonintDfmla = nonint$
 $\langle proof \rangle$

end-of-context GM-sec-model-finite