Showing invariance compositionally for a process algebra for network protocols

Timothy Bourke[1,2] Robert J. van Glabbeek[3] Peter Höfner[3]

1. INRIA Paris-Rocquencourt
2. École normale supérieure (DI)
3. NICTA

16 July 2014, ITP, Vienna, Austria.

Specification and Verification of Reactive Systems

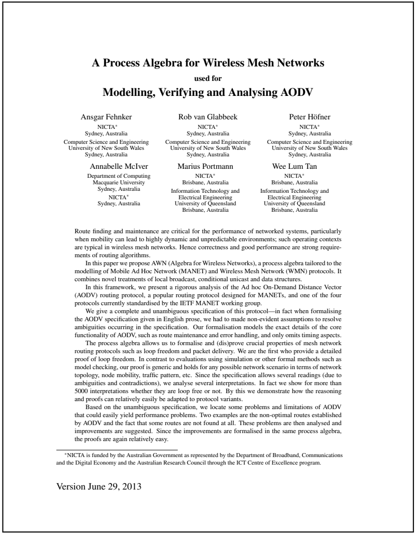
- Wireless network protocols
  (e.g., AODV routing protocol, RFC3561).
- Each network node is a reactive system.
- We prove properties of (arbitrary) networks of nodes.

- Modelling language: the process algebra AWN.
- Proof technique: inductive invariants (after Manna and Pnueli),
  plus 'open', lifting, and transfer rules.

Application of Isabelle/HOL

- Language definition and many proofs are standard.
- One or two tricks to mechanize.
- Informed by O. Müller's thesis work (in particular).

Pencil-and-paper model and proof



A Process Algebra for Wireless Mesh Networks

used for

Modelling, Verifying and Analysing AODV

Ansgar Fehnker
NICTA*
Sydney, Australia
Computer Science and Engineering
University of New South Wales
Sydney, Australia

Rob van Glabbeek
NICTA*
Sydney, Australia
Computer Science and Engineering
University of New South Wales
Sydney, Australia

Peter Höfner
NICTA*
Sydney, Australia
Computer Science and Engineering
University of New South Wales
Sydney, Australia

Annabelle McIver
Department of Computing
Macquarie University
Sydney, Australia
NICTA*
Sydney, Australia

Marius Portmann
NICTA*
Brisbane, Australia
Information Technology and
Electrical Engineering
University of Queensland
Brisbane, Australia

Wee Lum Tan
NICTA*
Brisbane, Australia
Information Technology and
Electrical Engineering
University of Queensland
Brisbane, Australia

Route finding and maintenance are critical for the performance of networked systems, particularly when mobility can lead to highly dynamic and unpredictable environments; such operating contexts are typical in wireless mesh networks. Hence correctness and good performance are strong requirements of routing algorithms.

In this paper we propose AWN (Algebra for Wireless Networks), a process algebra tailored to the modelling of Mobile Ad Hoc Network (MANET) and Wireless Mesh Network (WMN) protocols. It combines novel treatments of local broadcast, conditional unicast and data structures.

In this framework, we present a rigorous analysis of the Ad hoc On-Demand Distance Vector (AODV) routing protocol, a popular routing protocol designed for MANETs, and one of the four protocols currently standardised by the IETF MANET working group.

We give a complete and unambiguous specification of this protocol—in fact when formalising the AODV specification given in English prose, we had to made non-evident assumptions to resolve ambiguities occurring in the specification. Our formalisation models the exact details of the core functionality of AODV, such as route maintenance and error handling, and only omits timing aspects.

The process algebra allows us to formalise and (dis)prove crucial properties of mesh network routing protocols such as loop freedom and packet delivery. We are the first who provide a detailed proof of loop freedom. In contrast to evaluations using simulation or other formal methods such as model checking, our proof is generic and holds for any possible network scenario in terms of network topology, node mobility, traffic pattern, etc. Since the specification allows several readings (due to ambiguities and contradictions), we analyse several interpretations. In fact we show for more than 5000 interpretations whether they are loop free or not. By this we demonstrate how the reasoning and proofs can relatively easily be adapted to protocol variants.

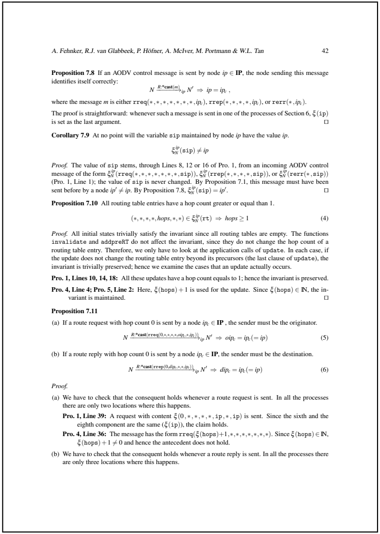
Based on the unambiguous specification, we locate some problems and limitations of AODV that could easily yield performance problems. Two examples are the non-optimal routes established by AODV and the fact that some routes are not found at all. These problems are then analysed and improvements are suggested. Since the improvements are formalised in the same process algebra, the proofs are again relatively easy.

*NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

Version June 29, 2013

- Team of experts in formal methods and wireless protocols.
- Layered process algebra AWN.

Pencil-and-paper model and proof



A. Fehnker, R.J. van Glabbeek, P. Höfner, A. McIver, M. Portmann & W.L. Tan 42

Proposition 7.8 If an AODV control message is sent by node $ip \in \mathbf{IP}$, the node sending this message identifies itself correctly:

$$N \xrightarrow{R:\mathbf{*cast}(m)}_{ip} N' \;\Rightarrow\; ip = ip_c\;,$$

where the message m is either $\mathtt{rreq}(*,*,*,*,*,*,*,ip_c)$, $\mathtt{rrep}(*,*,*,*,ip_c)$, or $\mathtt{rerr}(*,ip_c)$.

The proof is straightforward: whenever such a message is sent in one of the processes of Section 6, $\xi(\mathtt{ip})$ is set as the last argument. □

Corollary 7.9 At no point will the variable sip maintained by node *ip* have the value *ip*.

$$\xi_N^{ip}(\mathtt{sip}) \neq ip$$

Proof. The value of sip stems, through Lines 8, 12 or 16 of Pro. 1, from an incoming AODV control message of the form $\xi_N^{ip}(\mathtt{rreq}(*,*,*,*,*,*,*,\mathtt{sip}))$, $\xi_N^{ip}(\mathtt{rrep}(*,*,*,*,\mathtt{sip}))$, or $\xi_N^{ip}(\mathtt{rerr}(*,\mathtt{sip}))$ (Pro. 1, Line 1); the value of sip is never changed. By Proposition 7.1, this message must have been sent before by a node $ip' \neq ip$. By Proposition 7.8, $\xi_N^{ip}(\mathtt{sip}) = ip'$. □

Proposition 7.10 All routing table entries have a hop count greater or equal than 1.

$$(*,*,*,*,hops,*,*) \in \xi_N^{ip}(\mathtt{rt}) \;\Rightarrow\; hops \geq 1 \tag{4}$$

Proof. All initial states trivially satisfy the invariant since all routing tables are empty. The functions invalidate and addpreRT do not affect the invariant, since they do not change the hop count of a routing table entry. Therefore, we only have to look at the application calls of update. In each case, if the update does not change the routing table entry beyond its precursors (the last clause of update), the invariant is trivially preserved; hence we examine the cases that an update actually occurs.

Pro. 1, Lines 10, 14, 18: All these updates have a hop count equals to 1; hence the invariant is preserved.

Pro. 4, Line 4; Pro. 5, Line 2: Here, $\xi(\mathtt{hops})+1$ is used for the update. Since $\xi(\mathtt{hops}) \in \mathbb{N}$, the invariant is maintained. □

Proposition 7.11

(a) If a route request with hop count 0 is sent by a node $ip_c \in \mathbf{IP}$, the sender must be the originator.

$$N \xrightarrow{R:\mathbf{*cast}(\mathtt{rreq}(0,*,*,*,*,oip_c,*,ip_c))}_{ip} N' \;\Rightarrow\; oip_c = ip_c (= ip) \tag{5}$$

(b) If a route reply with hop count 0 is sent by a node $ip_c \in \mathbf{IP}$, the sender must be the destination.

$$N \xrightarrow{R:\mathbf{*cast}(\mathtt{rrep}(0,dip_c,*,*,ip_c))}_{ip} N' \;\Rightarrow\; dip_c = ip_c (= ip) \tag{6}$$

Proof.

(a) We have to check that the consequent holds whenever a route request is sent. In all the processes there are only two locations where this happens.

Pro. 1, Line 39: A request with content $\xi(0,*,*,*,*,\mathtt{ip},*,\mathtt{ip})$ is sent. Since the sixth and the eighth component are the same ($\xi(\mathtt{ip})$), the claim holds.

Pro. 4, Line 36: The message has the form $\mathtt{rreq}(\xi(\mathtt{hops})+1,*,*,*,*,*,*,*)$. Since $\xi(\mathtt{hops}) \in \mathbb{N}$, $\xi(\mathtt{hops})+1 \neq 0$ and hence the antecedent does not hold.

(b) We have to check that the consequent holds whenever a route reply is sent. In all the processes there are only three locations where this happens.

- Team of experts in formal methods and wireless protocols.
- Layered process algebra AWN.

Invariants

- Fastidious proofs over nodes.
- Looser extension to networks.

Outline

Modelling (AWN)

Proof

- Basic proof
- Open proof
- Lifting and transfer

Conclusion

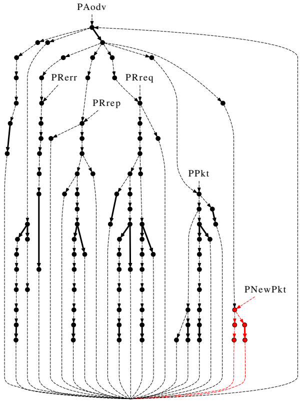
Modelling Network Protocols

	description	*state*
protocol	recursive specifications: Γ	pairs: (ξ, p) deep embedding for terms shallow embedding for data
networks	terms: $\langle D; \{A\}\rangle$, $\_ \parallel \_$.	trees of tuples

Modelling Network Protocols

	description	*state*
protocol	recursive specifications: Γ	pairs: (ξ, p) deep embedding for terms shallow embedding for data
networks	terms: $\langle D; \{A\}\rangle$, $\_ \parallel \_$.	trees of tuples

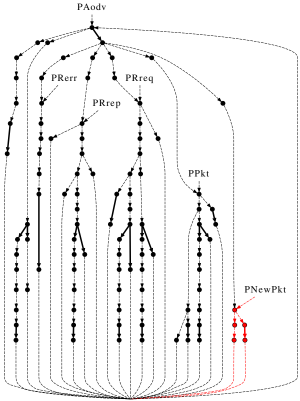
Γ_{AODV} PNewPkt $=$ labelled PNewPkt (
$\langle \lambda\xi.$ if dip $\xi =$ ip ξ then $\{\xi\}$ else $\emptyset\rangle$
deliver(data) . $[\![$clear-locals$]\!]$ call(PAodv)
$\oplus$
$\langle \lambda\xi.$ if dip $\xi \neq$ ip ξ then $\{\xi\}$ else $\emptyset\rangle$
$[\![\lambda\xi.\ \xi(\!|$store $:=$ add (data ξ) (dip ξ) (store ξ)$|\!)]\!]$
$[\![$clear-locals$]\!]$ call(PAodv))



Modelling Network Protocols

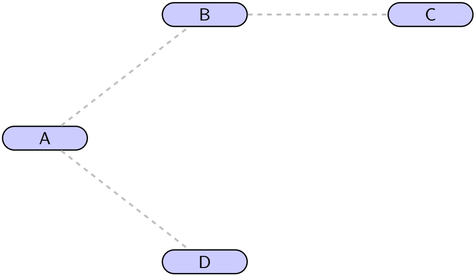
	description	*state*
protocol	recursive specifications: Γ	pairs: (ξ, p) deep embedding for terms shallow embedding for data
networks	terms: $\langle D; \{A\}\rangle$, $\_ \parallel \_$.	trees of tuples

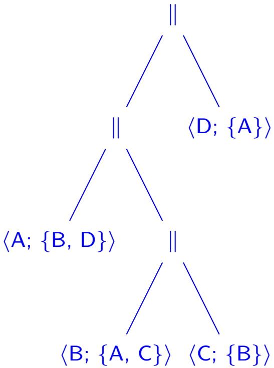
```
record state =
  ip     :: "ip"
  sn     :: "sqn"
  rt     :: "rt"
  rreqs :: "(ip × rreqid) set"
  store :: "store"

  msg      :: "msg"
  data     :: "data"
  dests   :: "ip ⇀ sqn"
  pre      :: "ip set"
  rreqid :: "rreqid"
  dip      :: "ip"
  oip      :: "ip"
  hops    :: "nat"
  ...
```



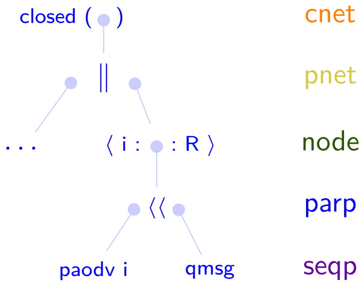
Modelling Network Protocols

	description	*state*
protocol	recursive specifications: Γ	pairs: (ξ, p) deep embedding for terms shallow embedding for data
networks	terms: $\langle D; \{A\}\rangle$, $\_ \parallel \_$.	trees of tuples





Mechanization of AWN

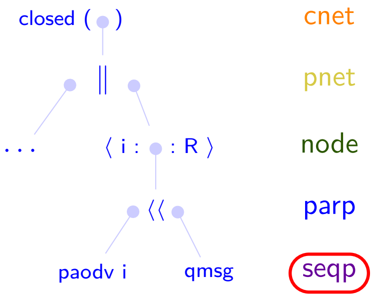


- AWN: layered process algebra
- SOS rules for each 'operator'
- Layers transform lower layers

- Model all as automata
  (initial states and transitions)

(|**init** :: 's set, **trans** :: ('s × 'a × 's) set |) :: ('s, 'a) automaton

Mechanization of AWN



- AWN: layered process algebra
- SOS rules for each ‘operator’
- Layers transform lower layers

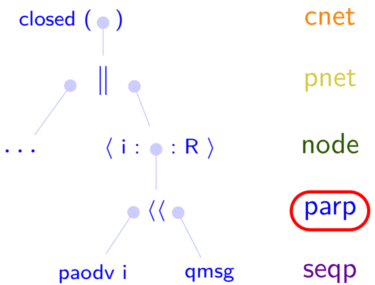
- Model all as automata
  (initial states and transitions)

$$\text{paodv i} = (\!|\text{init} = \{(\text{aodv-init i}, \Gamma_{\text{AODV}}\ \text{PAodv})\},\ \text{trans} = \text{seqp-sos}\ \Gamma_{\text{AODV}}|\!)$$

$$\frac{\xi' = \text{fa}\ \xi}{((\xi, \{l\}[\![\text{fa}]\!]\ p), \tau, (\xi', p)) \in \text{seqp-sos}\ \Gamma} \qquad \frac{((\xi, \Gamma\ \text{pn}), a, (\xi', p')) \in \text{seqp-sos}\ \Gamma}{((\xi, \text{call(pn)}), a, (\xi', p')) \in \text{seqp-sos}\ \Gamma}$$

$$((\xi, \{l\}\text{groupcast(ips, ms)} . p), \text{groupcast (ips}\ \xi\text{) (ms}\ \xi\text{)}, (\xi, p)) \in \text{seqp-sos}\ \Gamma$$

Mechanization of AWN



- AWN: layered process algebra
- SOS rules for each ‘operator’
- Layers transform lower layers

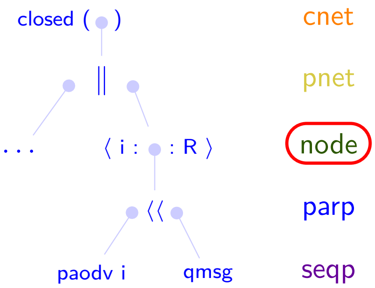
- Model all as automata
(initial states and transitions)

$$\mathsf{s} \langle\!\langle \mathsf{t} \equiv (\!|\mathsf{init} = \mathsf{init\ s} \times \mathsf{init\ t},\ \mathsf{trans} = \mathsf{parp\text{-}sos}\ (\mathsf{trans\ s})\ (\mathsf{trans\ t})|\!)$$

$$\frac{(\mathsf{s}, \mathsf{a}, \mathsf{s'}) \in \mathsf{S} \qquad \bigwedge \mathsf{m}.\ \mathsf{a} \neq \mathsf{receive\ m}}{((\mathsf{s}, \mathsf{t}), \mathsf{a}, (\mathsf{s'}, \mathsf{t})) \in \mathsf{parp\text{-}sos\ S\ T}} \qquad \frac{(\mathsf{t}, \mathsf{a}, \mathsf{t'}) \in \mathsf{T} \qquad \bigwedge \mathsf{m}.\ \mathsf{a} \neq \mathsf{send\ m}}{((\mathsf{s}, \mathsf{t}), \mathsf{a}, (\mathsf{s}, \mathsf{t'})) \in \mathsf{parp\text{-}sos\ S\ T}}$$

$$\frac{(\mathsf{s}, \mathsf{receive\ m}, \mathsf{s'}) \in \mathsf{S} \qquad (\mathsf{t}, \mathsf{send\ m}, \mathsf{t'}) \in \mathsf{T}}{((\mathsf{s}, \mathsf{t}), \tau, (\mathsf{s'}, \mathsf{t'})) \in \mathsf{parp\text{-}sos\ S\ T}}$$

Mechanization of AWN



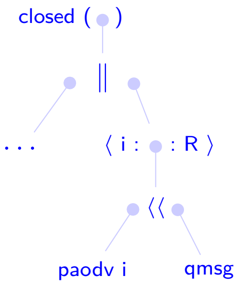
- AWN: layered process algebra
- SOS rules for each 'operator'
- Layers transform lower layers

- Model all as automata
  (initial states and transitions)

$$\langle i : S : R \rangle \equiv (\!|\,\text{init} = \{s^{i}_{R} \mid s \in \text{init } S\},\ \text{trans} = \text{node-sos } (\text{trans } S)\,|\!)$$

$$\frac{(s,\ \text{groupcast } D\ m,\ s') \in S}{(s^{i}_{R},\ (R \cap D)\text{:*cast}(m),\ s'^{i}_{R}) \in \text{node-sos } S} \qquad (P^{i}_{R},\ \text{connect}(i,\ i'),\ P^{i}_{R\,\cup\,\{i'\}}) \in \text{node-sos } S$$

Mechanization of AWN



cnet

pnet

node

parp

seqp

- AWN: layered process algebra
- SOS rules for each ‘operator’
- Layers transform lower layers

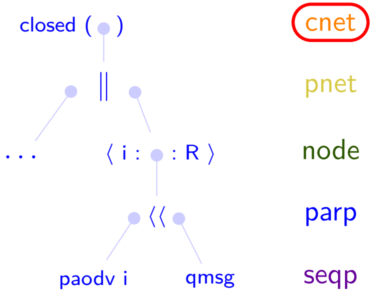
- Model all as automata
  (initial states and transitions)

$$
\begin{aligned}
\text{pnet np } \langle \text{i}; \text{R} \rangle &\quad = \quad \langle \text{i : np i : R} \rangle \\
\text{pnet np } (\text{p}_1 \parallel \text{p}_2) &\quad = \quad (\!|\text{init} = \{\text{s1} \shortparallel \text{s2} \mid \text{s1} \in \text{init (pnet np p}_1) \wedge \text{s2} \in \text{init (pnet np p}_2)\}, \\
&\qquad\quad \text{trans} = \text{pnet-sos (trans (pnet np p}_1)) \text{ (trans (pnet np p}_2))|\!)
\end{aligned}
$$

$$
\frac{(\text{s}, \tau, \text{s'}) \in \text{S}}{(\text{s} \shortparallel \text{t}, \tau, \text{s'} \shortparallel \text{t}) \in \text{pnet-sos S T}}
$$

$$
\frac{(\text{s, R:*cast(m), s'}) \in \text{S} \qquad (\text{t, H}\neg\text{K:arrive(m), t'}) \in \text{T} \qquad \text{H} \subseteq \text{R} \qquad \text{K} \cap \text{R} = \emptyset}{(\text{s} \shortparallel \text{t, R:*cast(m), s'} \shortparallel \text{t'}) \in \text{pnet-sos S T}}
$$

Mechanization of AWN



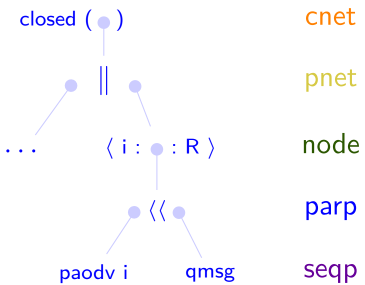
- AWN: layered process algebra
- SOS rules for each ‘operator’
- Layers transform lower layers

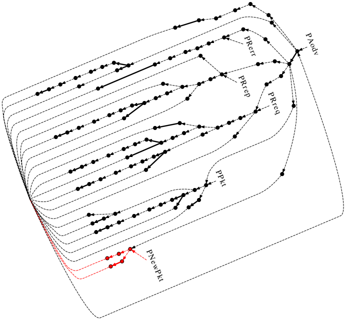
- Model all as automata (initial states and transitions)

closed A = A(|trans := cnet-sos (trans A)|)

(no receives without corresponding sends)

Mechanization of AWN





Outline

Stating invariant properties

Reachability

$$\frac{\mathsf{s} \in \mathsf{init}\ \mathsf{A}}{\mathsf{s} \in \mathsf{reachable}\ \mathsf{A}\ \mathsf{I}} \qquad \frac{\mathsf{s} \in \mathsf{reachable}\ \mathsf{A}\ \mathsf{I} \quad (\mathsf{s},\ \mathsf{a},\ \mathsf{s'}) \in \mathsf{trans}\ \mathsf{A} \quad \mathsf{I}\ \mathsf{a}}{\mathsf{s'} \in \mathsf{reachable}\ \mathsf{A}\ \mathsf{I}}$$

- Focus on invariants of states and steps.
- Not necessary to reason over traces.
- Different approach to the original proof.

Invariants

$\mathsf{A} \Vdash (\mathsf{I} \rightarrow)\ \mathsf{P} = \forall \mathsf{s} \in \mathsf{reachable}\ \mathsf{A}\ \mathsf{I}.\ \mathsf{P}\ \mathsf{s}$

Step Invariants

$\mathsf{A} \Vvdash (\mathsf{I} \rightarrow)\ \mathsf{P} = \forall \mathsf{a}.\ \mathsf{I}\ \mathsf{a} \rightarrow (\forall \mathsf{s} \in \mathsf{reachable}\ \mathsf{A}\ \mathsf{I}.\ \forall \mathsf{s'}.\ (\mathsf{s}, \mathsf{a}, \mathsf{s'}) \in \mathsf{trans}\ \mathsf{A} \rightarrow \mathsf{P}\ (\mathsf{s}, \mathsf{a}, \mathsf{s'}))$

(Invariant) Proof Strategy

cnet-sos

pnet-sos

node-sos

parp-sos

seqp-sos

(Invariant) Proof Strategy

cnet-sos closed (pnet (λi. paodv i $\langle\langle$ qmsg) n) $\Vert\!\!\models$ P

pnet-sos

node-sos

parp-sos

seqp-sos

(Invariant) Proof Strategy

cnet-sos closed (pnet (λi. paodv i ⟪ qmsg) n) $\Vert\!\!\models$ P

pnet-sos

node-sos

parp-sos

seqp-sos paodv i $\Vert\!\!\models$ P

(Invariant) Proof Strategy

cnet-sos	closed (pnet (λi. paodv i $\langle\langle$ qmsg) n) $\VDash$ P
pnet-sos	
node-sos	
parp-sos	paodv i $\langle\langle$ qmsg $\VDash$ P
$\uparrow$ lift	
seqp-sos	paodv i $\VDash$ P

(Invariant) Proof Strategy

cnet-sos	closed (pnet (λi. paodv i $\langle\langle$ qmsg) n) $\Vdash$ P
pnet-sos	
node-sos	$\langle$i : paodv i $\langle\langle$ qmsg : $\mathrm{R}_i\rangle$ $\Vdash$ P
↑ lift	
parp-sos	paodv i $\langle\langle$ qmsg $\Vdash$ P
↑ lift	
seqp-sos	paodv i $\Vdash$ P

(Invariant) Proof Strategy

cnet-sos	$\text{closed (pnet } (\lambda\text{i. paodv i} \langle\langle \text{ qmsg) n)} \VDash P$
pnet-sos	$\text{pnet } (\lambda\text{i. paodv i} \langle\langle \text{ qmsg) n} \VDash P$
↑ lift	
node-sos	$\langle \text{i : paodv i} \langle\langle \text{ qmsg : R}_i \rangle \VDash P$
↑ lift	
parp-sos	$\text{paodv i} \langle\langle \text{ qmsg} \VDash P$
↑ lift	
seqp-sos	$\text{paodv i} \VDash P$

(Invariant) Proof Strategy

cnet-sos — closed (pnet (λi. paodv i $\langle\!\langle$ qmsg) n) $\Vvdash P$

↑ lift

pnet-sos — pnet (λi. paodv i $\langle\!\langle$ qmsg) n $\Vvdash P$

↑ lift

node-sos — $\langle$i : paodv i $\langle\!\langle$ qmsg : $\mathrm{R}_i\rangle \Vvdash P$

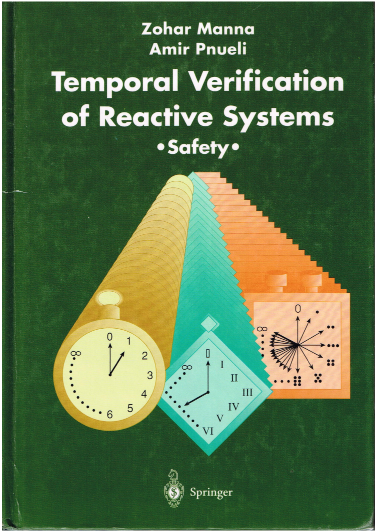
↑ lift

parp-sos — paodv i $\langle\!\langle$ qmsg $\Vvdash P$

↑ lift

seqp-sos — paodv i $\Vvdash P$

Verifying safety properties of reactive systems



- Published in 1995. Companion to *The Temporal Logic of Reactive and Concurrent Systems: Specification*
- Existing theory enough for (most of) the invariants over individual processes (Floyd's inductive invariants)
- *vs* TLA+, I/O Automata, Paulson's inductive method...
- Temporal logic formulas as 'proof patterns' of which we only need one...

The basic ‘pattern’ for showing invariance

For an assertion φ,

$$\frac{\begin{array}{ll}\text{B1.} & \Theta \rightarrow \varphi \\ \text{B2.} & \{\varphi\}\ \mathcal{T}\ \{\varphi\}\end{array}}{\Box\varphi}$$

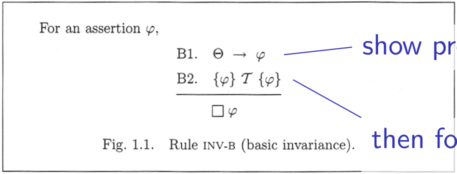
Fig. 1.1. Rule INV-B (basic invariance).

show property of initial states

then for every transition:

- assume the property of the pre state (φ)
- show the property of the post state (φ')

The basic ‘pattern’ for showing invariance



For an assertion φ,

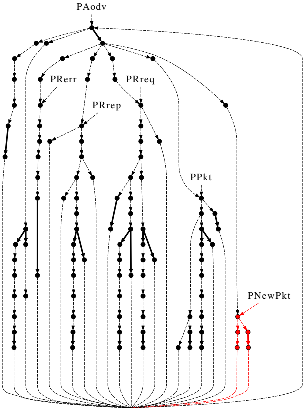
$$\begin{array}{ll} \text{B1.} & \Theta \rightarrow \varphi \\ \text{B2.} & \{\varphi\}\ \mathcal{T}\ \{\varphi\} \\ \hline & \Box\varphi \end{array}$$

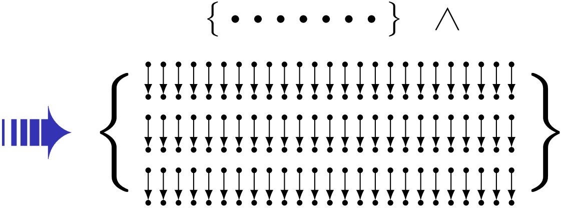
Fig. 1.1. Rule INV-B (basic invariance).

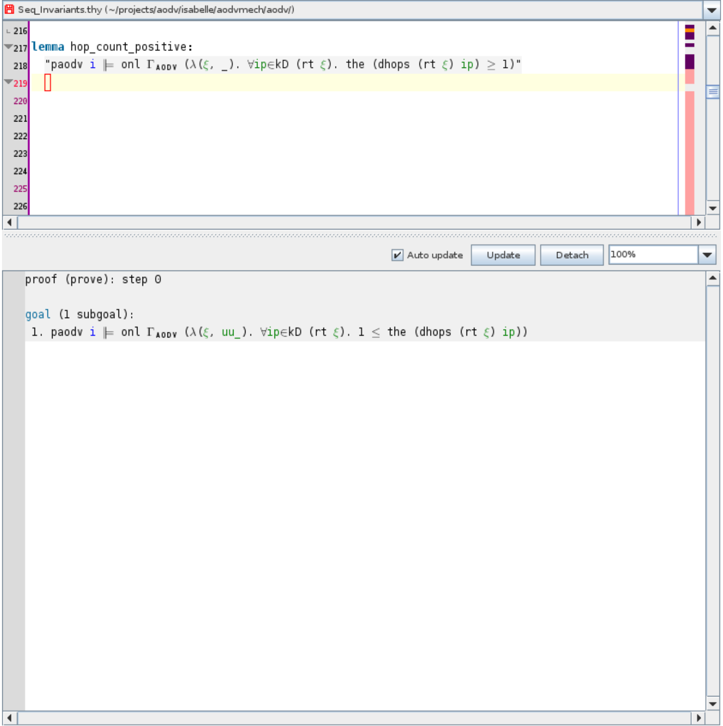
show property of initial states

then for every transition:

- assume the property of the pre state (φ)
- show the property of the post state (φ')






Seq_Invariants.thy (~/projects/aodv/isabelle/aodvmech/aodv/)
lemma hop_count_positive:
"paodv i ⊫ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
Auto update
Update
Detach
100%
proof (prove): step 0
goal (1 subgoal):
1. paodv i ⊫ onl Γ_AODV (λ(ξ, uu_). ∀ip∈kD (rt ξ). 1 ≤ the (dhops (rt ξ) ip))

Seq_Invariants.thy (~/projects/aodv/isabelle/aodvmech/aodv/)

```
lemma hop_count_positive:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  apply (inv_cterms (intro_only))
```

Auto update | Update | Detach | 100%

```
proof (prove): step 1

goal (2 subgoals):
 1. ⋀ξ p l.
       (ξ, p) ∈ init (paodv i) ⟹
       l ∈ labels Γ_AODV p ⟹ case (ξ, l) of (ξ, uu_) ⇒ ∀ip∈kD (rt ξ). 1 ≤ the (dhops (rt ξ) ip)
 2. ⋀p l ξ a q l' ξ' pp pn.
       wellformed Γ_AODV ⟹
       p ∈ ctermsl (Γ_AODV pn) ⟹
       not_call p ⟹
       l ∈ labels Γ_AODV p ⟹
       case (ξ, l) of (ξ, uu_) ⇒ ∀ip∈kD (rt ξ). 1 ≤ the (dhops (rt ξ) ip) ⟹
       ((ξ, p), a, ξ', q) ∈ seqp_sos Γ_AODV ⟹
       ((ξ, p), a, ξ', q) ∈ automaton.trans (paodv i) ⟹
       l' ∈ labels Γ_AODV q ⟹
       (ξ, pp) ∈ reachable (paodv i) TT ⟹
       p ∈ sterms Γ_AODV pp ⟹
       (ξ', q) ∈ reachable (paodv i) TT ⟹
       TT a ⟹ case (ξ', l') of (ξ, uu_) ⇒ ∀ip∈kD (rt ξ). 1 ≤ the (dhops (rt ξ) ip)
```

Seq_Invariants.thy (~/projects/aodv/isabelle/aodvmech/aodv/)

```
lemma hop_count_positive:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  apply (inv_cterms (vcs_only))
```

Auto update | Update | Detach | 100%

```
proof (prove): step 1

goal (103 subgoals):
 1. ⋀p l ξ a q l' ξ' pp pn p' dip.
       wellformed Γ_AODV ⟹
       not_call
        ({PAodv-:0}⟨λξ. {ξ⦇dip := dip⦈ |dip. dip ∈ qD (store ξ) ∧ dip ∉ vD (rt ξ) ∧ the σ↘p-flag↙(store ξ,
         p') ⟹
       l ∈ {PAodv-:0} ⟹
       ∀ip∈kD (rt ξ). 1 ≤ the (dhops (rt ξ) ip) ⟹
       ((ξ, {PAodv-:0}⟨λξ. {ξ⦇dip := dip⦈ |dip. dip ∈ qD (store ξ) ∧ dip ∉ vD (rt ξ) ∧ the σ↘p-flag↙(store
         p'),
        τ, ξ', p')
       ∈ automaton.trans (paodv i) ⟹
       l' ∈ labels Γ_AODV p' ⟹
       (ξ, pp) ∈ reachable (paodv i) TT ⟹
       {PAodv-:0}⟨λξ. {ξ⦇dip := dip⦈ |dip. dip ∈ qD (store ξ) ∧ dip ∉ vD (rt ξ) ∧ the σ↘p-flag↙(store ξ, d
       p' ∈ sterms Γ_AODV pp ⟹
       (ξ', p') ∈ reachable (paodv i) TT ⟹
       TT τ ⟹
       p = {PAodv-:0}⟨λξ. {ξ⦇dip := dip⦈ |dip. dip ∈ qD (store ξ) ∧ dip ∉ vD (rt ξ) ∧ the σ↘p-flag↙(store
       p' ⟹
       l' ∈ {PAodv-:21} ⟹
       a = τ ⟹
       q = p' ⟹
```

Seq_Invariants.thy (~/projects/aodv/isabelle/aodvmech/aodv/)

```
lemma hop_count_positive:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  apply (inv_cterms)
```

Auto update | Update | Detach | 100%

```
proof (prove): step 1

goal (9 subgoals):
 1. ⋀p l ξ a q l' ξ' pp p'.
       l = PAodv-:8 ⟹
       ∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip) ⟹
       ((ξ, {PAodv-:8}⟦λξ. ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈⟧
         p'),
        τs, ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈, p')
       ∈ seqp_sos Γ_AODV ⟹
       PAodv-:9 ∈ labels Γ_AODV p' ⟹
       (ξ, pp) ∈ reachable (paodv i) TT ⟹
       {PAodv-:8}⟦λξ. ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈⟧
       p' ∈ sterms Γ_AODV pp ⟹
       (ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈, p') ∈ reachable (paodv i) TT ⟹
       p = {PAodv-:8}⟦λξ. ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈⟧
       p' ⟹
       l' = PAodv-:9 ⟹
       a = τs ⟹
       ξ' = ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈ ⟹
       q = p' ⟹
       Suc 0 ≤ the (dhops (update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})) (sip ξ)) ∧
       (∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})) ip))
 2. ⋀p l ξ a q l' ξ' pp p'.
       l = PAodv-:6 ⟹
```

Seq_Invariants.thy (~/projects/aodv/isabelle/aodvmech/aodv/)

```
lemma hop_count_positive:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  apply (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined])
```

Auto update | Update | Detach | 100%

```
proof (prove): step 1

goal (5 subgoals):
 1. ⋀p l ξ a q l' ξ' pp p'.
       l = PAodv-:8 ⟹
       ∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (rt ξ) ip) ⟹
       ((ξ, {PAodv-:8}⟦λξ. ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈⟧
         p'),
        τs, ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈, p')
       ∈ seqp_sos Γ_AODV ⟹
       PAodv-:9 ∈ labels Γ_AODV p' ⟹
       (ξ, pp) ∈ reachable (paodv i) TT ⟹
       {PAodv-:8}⟦λξ. ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈⟧
       p' ∈ sterms Γ_AODV pp ⟹
       (ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈, p') ∈ reachable (paodv i) TT ⟹
       p = {PAodv-:8}⟦λξ. ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈⟧
       p' ⟹
       l' = PAodv-:9 ⟹
       a = τs ⟹
       ξ' = ξ⦇rt := update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})⦈ ⟹
       q = p' ⟹
       Suc 0 ≤ the (dhops (update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})) (sip ξ)) ∧
       (∀ip∈kD (rt ξ). Suc 0 ≤ the (dhops (update (rt ξ) (sip ξ) (0, unk, val, Suc 0, sip ξ, {})) ip))
 2. ⋀p l ξ a q l' ξ' pp p'.
       l = PAodv-:6 ⟹
```

Seq_Invariants.thy (~/projects/aodv/isabelle/aodvmech/aodv/)

```
lemma hop_count_positive:
  "paodv i ⊫ onl Γ_AODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
  apply (inv_cterms inv add: onl_invariant_sterms [OF aodv_wf addpreRT_welldefined])
  apply auto
```

Auto update | Update | Detach | 100%

```
proof (prove): step 2

goal:
No subgoals!
```

The problem with global invariants

Theorem 7.29 The quality of the routing table entries for a destination *dip* is strictly increasing along a route towards *dip*, until it reaches either *dip* or a node with an invalided routing table entry to *dip*.

$$dip \in \mathtt{vD}_N^{ip} \cap \mathtt{vD}_N^{nhip} \wedge nhip \neq dip \Rightarrow \xi_N^{ip}(\mathtt{rt}) \sqsubset_{dip} \xi_N^{nhip}(\mathtt{rt})\ , \quad (21)$$

where N is a reachable network expression and $nhip := \mathtt{nhop}_N^{ip}(dip)$ is the IP address of the next hop.

The problem with global invariants

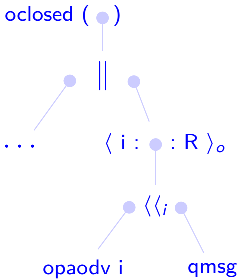
Theorem 7.29 The quality of the routing table entries for a destination *dip* is strictly increasing along a route towards *dip*, until it reaches either *dip* or a node with an invalided routing table entry to *dip*.

$$dip \in \mathtt{vD}_N^{ip} \cap \mathtt{vD}_N^{nhip} \wedge nhip \neq dip \;\Rightarrow\; \xi_N^{ip}(\mathtt{rt}) \sqsubset_{dip} \xi_N^{nhip}(\mathtt{rt}) \,, \tag{21}$$

where N is a reachable network expression and $nhip := \mathtt{nhop}_N^{ip}(dip)$ is the IP address of the next hop.

- We must state a property of routing tables across pairs of nodes, i.e., elements of a global state
- ...that does not exist at the level of individual sequential processes.

An ‘open model’ of AWN



ocnet

opnet

onode

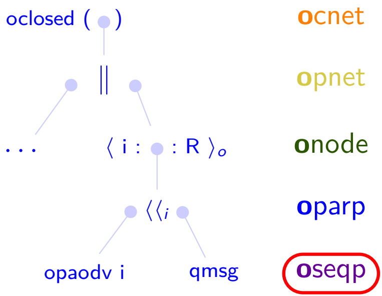
oparp

oseqp

$$\xi :: \text{state}$$

$$\sigma :: \text{ip} \Rightarrow \text{state}$$

An 'open model' of AWN



ξ :: state

σ :: ip $\Rightarrow$ state

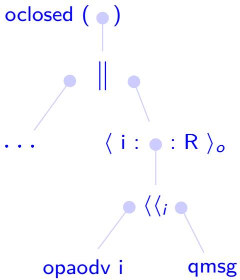
opaodv i $=$ (|init $= \{$(aodv-init, Γ_{AODV} PAodv)$\}$, trans $=$ oseqp-sos Γ_{AODV} i|).

$$\frac{\xi' = \text{fa } \xi}{((\xi,\ \{l\}[\![\text{fa}]\!]\ p),\ \tau,\ (\xi',\ p)) \in \text{seqp-sos } \Gamma}$$

versus

$$\frac{\sigma'\ i = \text{fa } (\sigma\ i)}{((\sigma,\ \{l\}[\![\text{fa}]\!]\ p),\ \tau,\ (\sigma',\ p)) \in \text{oseqp-sos } \Gamma\ i}$$

An ‘open model’ of AWN



ocnet

opnet

onode

oparp

oseqp

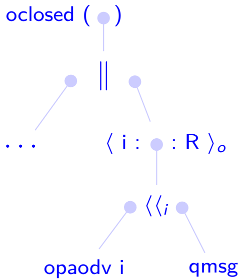
ξ :: state

σ :: ip $\Rightarrow$ state

$$\frac{((\sigma,\ P),\ \text{groupcast D m},\ \sigma',\ P') \in S}{((\sigma,\ P_R^i),\ (R \cap D)\text{:*cast(m)},\ (\sigma',\ P'^{\,i}_R)) \in \textbf{o}\text{node-sos } S}$$

$$\frac{((\sigma,\ P),\ \tau,\ (\sigma',\ P')) \in S \qquad \forall j \neq i.\ \sigma'\ j = \sigma\ j}{((\sigma,\ P_R^i),\ \tau,\ (\sigma',\ P'^{\,i}_R)) \in \textbf{o}\text{node-sos } S}$$

An 'open model' of AWN



ocnet

opnet

onode

oparp

oseqp

ξ :: state

σ :: ip $\Rightarrow$ state

$$
\begin{aligned}
\text{opnet np } \langle \text{i; R} \rangle \quad &= \quad \langle \text{i : np i : R} \rangle_o \\
\text{opnet np } (\text{p}_1 \parallel \text{p}_2) \quad &= \quad (\!|\text{init} = \{(\sigma, \text{s1} \shortparallel \text{s2}) \mid (\sigma, \text{s1}) \in \text{init (opnet np p}_1) \\
&\qquad\qquad \wedge (\sigma, \text{s2}) \in \text{init (opnet np p}_2) \\
&\qquad\qquad \wedge \text{net-ips s1} \cap \text{net-ips s2} = \emptyset\}, \\
&\qquad \text{trans} = \text{opnet-sos (trans (opnet np p}_1)) \text{ (trans (opnet np p}_2))|\!)
\end{aligned}
$$

$$
\frac{((\sigma, \text{s}), \text{H}\neg\text{K:arrive(m)}, (\sigma', \text{s}')) \in \text{S} \qquad ((\sigma, \text{t}), \text{H}'\neg\text{K}'\text{:arrive(m)}, (\sigma', \text{t}')) \in \text{T}}{((\sigma, \text{s} \shortparallel \text{t}), (\text{H} \cup \text{H}')\neg(\text{K} \cup \text{K}')\text{:arrive(m)}, (\sigma', \text{s}' \shortparallel \text{t}')) \in \text{opnet-sos S T}}
$$

Open invariants

Open reachability

$$\frac{(\sigma,\ p)\ \in\ \text{init A}}{(\sigma,\ p)\ \in\ \text{oreachable A S U}} \qquad \frac{(\sigma,\ p)\ \in\ \text{oreachable A S U} \qquad \text{U}\ \sigma\ \sigma'}{(\sigma',\ p)\ \in\ \text{oreachable A S U}}$$

$$\frac{(\sigma,\ p)\ \in\ \text{oreachable A S U} \qquad ((\sigma,\ p),\ a,\ (\sigma',\ p'))\ \in\ \text{trans A} \qquad \text{S}\ \sigma\ \sigma'\ a}{(\sigma',\ p')\ \in\ \text{oreachable A S U}}$$

Open invariants

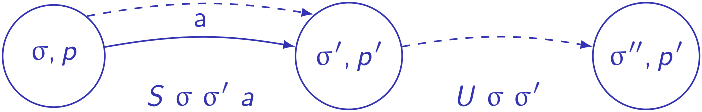
Open reachability

interleaving steps must satisfy U

$$\frac{(\sigma,\ p)\ \in\ \text{init A}}{(\sigma,\ p)\ \in\ \text{oreachable A S U}} \qquad \frac{(\sigma,\ p)\ \in\ \text{oreachable A S U} \qquad \text{U}\ \sigma\ \sigma'}{(\sigma',\ p)\ \in\ \text{oreachable A S U}}$$

$$\frac{(\sigma,\ p)\ \in\ \text{oreachable A S U} \qquad ((\sigma,\ p),\ \text{a},\ (\sigma',\ p'))\ \in\ \text{trans A} \qquad \text{S}\ \sigma\ \sigma'\ \text{a}}{(\sigma',\ p')\ \in\ \text{oreachable A S U}}$$

'local' steps must satisfy S



Open invariants

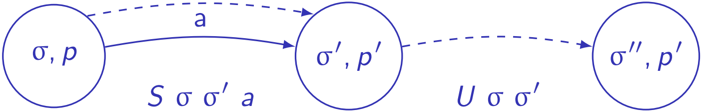
Open reachability

interleaving steps must satisfy U

$$\frac{(\sigma,\ p)\ \in\ \text{init A}}{(\sigma,\ p)\ \in\ \text{oreachable A S U}} \qquad \frac{(\sigma,\ p)\ \in\ \text{oreachable A S U} \qquad \text{U}\ \sigma\ \sigma'}{(\sigma',\ p)\ \in\ \text{oreachable A S U}}$$

$$\frac{(\sigma,\ p)\ \in\ \text{oreachable A S U} \qquad ((\sigma,\ p),\ \text{a},\ (\sigma',\ p'))\ \in\ \text{trans A} \qquad \text{S}\ \sigma\ \sigma'\ \text{a}}{(\sigma',\ p')\ \in\ \text{oreachable A S U}}$$

'local' steps must satisfy S



other P A σ σ' ≡ ∀i. if i ∈ A then σ' i = σ i else P (σ i) (σ' i)

otherwith P A I σ σ' a ≡ (∀i. i ∉ A → P (σ i) (σ' i)) ∧ I σ a

Open invariants

Open reachability

interleaving steps must satisfy *U*

$$\frac{(\sigma,\ p) \in \text{init A}}{(\sigma,\ p) \in \text{oreachable A S U}} \qquad \frac{(\sigma,\ p) \in \text{oreachable A S U} \qquad \text{U}\ \sigma\ \sigma'}{(\sigma',\ p) \in \text{oreachable A S U}}$$

$$\frac{(\sigma,\ p) \in \text{oreachable A S U} \qquad ((\sigma,\ p),\ a,\ (\sigma',\ p')) \in \text{trans A} \qquad \text{S}\ \sigma\ \sigma'\ a}{(\sigma',\ p') \in \text{oreachable A S U}}$$

'local' steps must satisfy *S*

Open Invariants

$\text{A} \models (\text{S, U} \rightarrow)\ \text{P} = \forall s \in \text{oreachable A S U. P s}$

Open Step Invariants

$\text{A} \models\!\!\!\equiv (\text{S, U} \rightarrow)\ \text{P} =$

$\quad \forall s \in \text{oreachable A S U}.\ \forall a\ s'.\ (s,\ a,\ s') \in \text{trans A} \wedge \text{S (fst s) (fst s') a} \rightarrow \text{P (s, a, s')}$

Open invariants

Open reachability

interleaving steps must satisfy U

$$\frac{(\sigma,\ p)\ \in\ \text{init A}}{(\sigma,\ p)\ \in\ \text{oreachable A S U}} \qquad \frac{(\sigma,\ p)\ \in\ \text{oreachable A S U} \qquad \text{U}\ \sigma\ \sigma'}{(\sigma',\ p)\ \in\ \text{oreachable A S U}}$$

$$\frac{(\sigma,\ p)\ \in\ \text{oreachable A S U} \qquad ((\sigma,\ p),\ a,\ (\sigma',\ p'))\ \in\ \text{trans A} \qquad \text{S}\ \sigma\ \sigma'\ a}{(\sigma',\ p')\ \in\ \text{oreachable A S U}}$$

'local' steps must satisfy S

Open Invariants

$\text{A} \models (\text{S},\ \text{U} \rightarrow)\ \text{P} = \forall s \in \text{oreachable A S U}.\ \text{P s}$

Open Step Invariants

$\text{A} \Vdash\!\equiv (\text{S},\ \text{U} \rightarrow)\ \text{P} =$

$\quad \forall s \in \text{oreachable A S U}.\ \forall a\ s'.\ (s,\ a,\ s')\ \in\ \text{trans A} \wedge \text{S (fst s) (fst s') a} \rightarrow \text{P}\ (s,\ a,\ s')$

Lift standard invariants

$$\frac{\text{initiali i (init OA) (init A)} \qquad \begin{array}{c}\text{A} \Vdash\!=_A (\text{I} \rightarrow)\ \text{P} \\ \text{trans OA} = \text{oseqp-sos}\ \Gamma\ \text{i}\end{array} \qquad \text{trans A} = \text{seqp-sos}\ \Gamma}{\text{OA} \models_A (\text{act I, other ANY \{i\}} \rightarrow)\ \text{seqll i P}}$$

Open invariants: proof rule (**o**seqp)

To prove the invariant $A \models (S, U \rightarrow) \text{ onl } \Gamma \; P$

1. Show for the initial states.

2. Show across each control term.

where

wellformed Γ

simple-labels Γ

control-within Γ (init A)

trans A = seqp-sos Γ

Open invariants: proof rule (**o**seqp)

To prove the invariant $A \models (S, U \rightarrow)$ onl Γ P

1. Show for the initial states.

2. Show across each control term.

3. Show for environment steps:

where wellformed Γ

simple-labels Γ

control-within Γ (init A)

trans A = seqp-sos Γ

assume:	$(\sigma, p) \in$ oreachable A S U	*in any oreachable state*
	$l \in$ labels Γ p	
	$P(\sigma, l)$	*assume the property is true*
	$U\ \sigma\ \sigma'$	*then, for all valid environment steps...*
show:	$P(\sigma', l)$	*...show that the property is preserved*

Outline

Lifting and transfer

cnet-sos

closed (pnet (λi. paodv i $\langle\langle_i$ qmsg) n) $\Vert\!\!\models$ P

pnet-sos

node-sos

parp-sos

seqp-sos

Lifting and transfer

cnet-sos

closed (pnet (λi. paodv i $\langle\langle_i$ qmsg) n) $\Vdash$ P

ocnet-sos

pnet-sos

opnet-sos

node-sos

onode-sos

parp-sos

oparp-sos

seqp-sos

oseqp-sos

Lifting and transfer

cnet-sos

closed (pnet (λi. paodv i $\langle\langle_i$ qmsg) n) $\Vdash P$

ocnet-sos

pnet-sos

opnet-sos

node-sos

onode-sos

parp-sos

oparp-sos

seqp-sos

oseqp-sos

opaodv i $\Vdash P_1'$

Lifting and transfer

cnet-sos | **o**cnet-sos

closed (pnet (λi. paodv i $\langle\langle_i$ qmsg) n) $\Vdash P$

pnet-sos | **o**pnet-sos

node-sos | **o**node-sos

parp-sos | **o**parp-sos

opaodv i $\langle\langle_i$ qmsg $\Vdash P'_2$

lift

seqp-sos | **o**seqp-sos

opaodv i $\Vdash P'_1$

Lifting and transfer

cnet-sos

closed (pnet (λi. paodv i $\langle\langle_i$ qmsg) n) $\Vert\!\!\models P$

ocnet-sos

pnet-sos

opnet-sos

node-sos

onode-sos

$\langle$i : opaodv i $\langle\langle_i$ qmsg : $\mathsf{R}_i\rangle_o \Vert\!\!\models P_3'$

lift

parp-sos

oparp-sos

opaodv i $\langle\langle_i$ qmsg $\Vert\!\!\models P_2'$

lift

seqp-sos

oseqp-sos

opaodv i $\Vert\!\!\models P_1'$

Lifting and transfer

cnet-sos | **o**cnet-sos

closed (pnet (λi. paodv i $\langle\langle_i$ qmsg) n) $\Vert\!\!\models P$

pnet-sos | **o**opnet-sos

opnet (λi. opaodv i $\langle\langle_i$ qmsg) n $\Vert\!\!\models P'_4$

$\uparrow$ lift

node-sos | **o**node-sos

$\langle$i : opaodv i $\langle\langle_i$ qmsg : $R_i\rangle_o$ $\Vert\!\!\models P'_3$

$\uparrow$ lift

parp-sos | **o**parp-sos

opaodv i $\langle\langle_i$ qmsg $\Vert\!\!\models P'_2$

$\uparrow$ lift

seqp-sos | **o**seqp-sos

opaodv i $\Vert\!\!\models P'_1$

Lifting and transfer

cnet-sos | **o**cnet-sos

closed (pnet (λi. paodv i $\langle\langle_i$ qmsg) n) $\Vdash P$

oclosed (opnet (λi. opaodv i $\langle\langle_i$ qmsg) n) $\Vdash P_5'$

lift

pnet-sos | **o**pnet-sos

opnet (λi. opaodv i $\langle\langle_i$ qmsg) n $\Vdash P_4'$

lift

node-sos | **o**node-sos

$\langle$i : opaodv i $\langle\langle_i$ qmsg : R$_i\rangle_o \Vdash P_3'$

lift

parp-sos | **o**parp-sos

opaodv i $\langle\langle_i$ qmsg $\Vdash P_2'$

lift

seqp-sos | **o**seqp-sos

opaodv i $\Vdash P_1'$

Lifting and transfer

transfer

cnet-sos ← ocnet-sos

closed (pnet (λi. paodv i $\langle\langle_i$ qmsg) n) $\Vert\!\!\models P$

oclosed (opnet (λi. opaodv i $\langle\langle_i$ qmsg) n) $\Vert\!\!\models P'_5$

lift

pnet-sos — opnet-sos

opnet (λi. opaodv i $\langle\langle_i$ qmsg) n $\Vert\!\!\models P'_4$

lift

node-sos — onode-sos

$\langle$i : opaodv i $\langle\langle_i$ qmsg : $R_i\rangle_o$ $\Vert\!\!\models P'_3$

lift

parp-sos — oparp-sos

opaodv i $\langle\langle_i$ qmsg $\Vert\!\!\models P'_2$

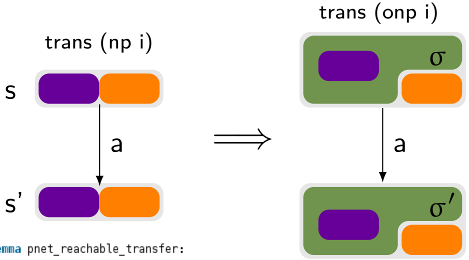
lift

seqp-sos — oseqp-sos

opaodv i $\Vert\!\!\models P'_1$

Transfer

```
locale openproc =
  fixes np  :: "ip ⇒ ('s, ('m::msg) seq_action) automaton"
    and onp :: "ip ⇒ ((ip ⇒ 'g) × 'l, 'm seq_action) automaton"
    and sr  :: "'s ⇒ ('g × 'l)"
  assumes  init: "{ (σ, ζ) |σ ζ s. s ∈ init (np i)
                              ∧ (σ i, ζ) = sr s
                              ∧ (∀j. j≠i ⟶ σ j ∈ (fst o sr) ` init (np j)) } ⊆ init (onp i)"
      and init_notempty: "∀j. init (np j) ≠ {}"
      and trans: "⋀s a s' σ σ'. ⟦ σ i = fst (sr s);
                                   σ' i = fst (sr s');
                                   (s, a, s') ∈ trans (np i) ⟧
                  ⟹ ((σ, snd (sr s)), a, (σ', snd (sr s'))) ∈ trans (onp i)"
```

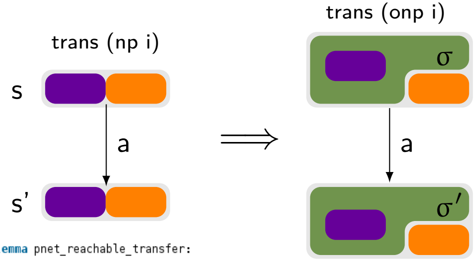



```
lemma pnet_reachable_transfer:
  assumes "wf_net_tree n"
      and "s ∈ reachable (closed (pnet np n)) TT"
    shows "initmissing (netgmap sr s) ∈ oreachable (oclosed (opnet onp n)) (λ_ _ _. True) U"

lemma close_opnet:
  assumes "wf_net_tree n"
      and "oclosed (opnet onp n) ⊫ (λ_ _ _. True, U →) global P"
    shows "closed (pnet np n) ⊫ netglobal P"
```

Transfer

```
locale openproc =
  fixes np  :: "ip ⇒ ('s, ('m::msg) seq_action) automaton"
    and onp :: "ip ⇒ ((ip ⇒ 'g) × 'l, 'm seq_action) automaton"
    and sr  :: "'s ⇒ ('g × 'l)"
  assumes  init: "{ (σ, ζ) |σ ζ s. s ∈ init (np i)
                             ∧ (σ i, ζ) = sr s
                             ∧ (∀j. j≠i ⟶ σ j ∈ (fst o sr) ` init (np j)) } ⊆ init (onp i)"
      and init_notempty: "∀j. init (np j) ≠ {}"
      and trans: "⋀s a s' σ σ'. ⟦ σ i = fst (sr s);
                                 σ' i = fst (sr s');
                                 (s, a, s') ∈ trans (np i) ⟧
                  ⟹ ((σ, snd (sr s)), a, (σ', snd (sr s'))) ∈ trans (onp i)"
```

- Instantiate with paodv/opaodv,
- and also with _ ⟪ qmsg

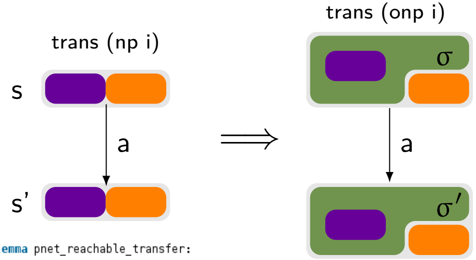



```
lemma pnet_reachable_transfer:
  assumes "wf_net_tree n"
      and "s ∈ reachable (closed (pnet np n)) TT"
    shows "initmissing (netgmap sr s) ∈ oreachable (oclosed (opnet onp n)) (λ_ _ _. True) U"

lemma close_opnet:
  assumes "wf_net_tree n"
      and "oclosed (opnet onp n) ⊨ (λ_ _ _. True, U →) global P"
    shows "closed (pnet np n) ⊫ netglobal P"
```

Transfer

```
locale openproc =
  fixes np  :: "ip ⇒ ('s, ('m::msg) seq_action) automaton"
    and onp :: "ip ⇒ ((ip ⇒ 'g) × 'l, 'm seq_action) automaton"
    and sr  :: "'s ⇒ ('g × 'l)"
  assumes  init: "{ (σ, ζ) |σ ζ s. s ∈ init (np i)
                              ∧ (σ i, ζ) = sr s
                              ∧ (∀j. j≠i ⟶ σ j ∈ (fst o sr) ` init (np j)) } ⊆ init (onp i)"
      and init_notempty: "∀j. init (np j) ≠ {}"
      and trans: "⋀s a s' σ σ'. ⟦ σ i = fst (sr s);
                                 σ' i = fst (sr s');
                                 (s, a, s') ∈ trans (np i) ⟧
                  ⟹ ((σ, snd (sr s)), a, (σ', snd (sr s'))) ∈ trans (onp i)"
```

- Instantiate with paodv/opaodv,
- and also with _ ⟪ qmsg



Lift from processes to networks

- Induction 'along' oreachable.
- Induction 'up' net_terms.
- Need to discharge 'assumptions' in rules.

```
lemma pnet_reachable_transfer:
  assumes "wf_net_tree n"
      and "s ∈ reachable (closed (pnet np n)) TT"
    shows "initmissing (netgmap sr s) ∈ oreachable (oclosed (opnet onp n)) (λ_ _ _. True) U"

lemma close_opnet:
  assumes "wf_net_tree n"
      and "oclosed (opnet onp n) ⊨ (λ_ _ _. True, U →) global P"
    shows "closed (pnet np n) ⊫ netglobal P"
```

Conclusion

- Framework for specifying and verifying a class of reactive systems.
- Compositional technique for stating and lifting (inductive) invariants.
- Applied to AODV (RFC3561)—coming soon.

- Beneficial to focus on a concrete verification task.
- No real process algebra.
  - More convenient than automaton transition tables.
  - The layered structure is important.

- Takes advantage of developments in and around Isabelle
  - PIDE, Isar, Locales,
  - Parallel proofs (parallel_goals), Poly/ML,
  - Sledgehammer, System on TPTP.