Channel Systems

Lecture #4 of Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling and Verification

E-mail: `katoen@cs.rwth-aachen.de`

October 29, 2008

Overview Lecture #4

- What is a channel system?
- Example: alternating bit protocol
- From channel systems to transition systems
- The modeling language `NanoPromela`
- Examples
- Semantics of `NanoPromela` models

Channels

- Processes communicate via *channels* ($c \in$ *Chan*)
- Channels are first-in, first-out buffers
- Channels are typed (wrt. their content — *dom*(c))
- Channels buffer messages (of appropriate type)
- Channel capacity = maximum $\#$ messages that can be stored
  - if *cap*$(c) \in \mathbb{N}$ then c is a channel with finite capacity
  - if *cap*$(c) = \infty$ then c has an infinite capacity
  - if *cap*$(c) > 0$, there is some “delay” between sending and receipt
  - if *cap*$(c) = 0$, then communication via c amounts to handshaking

Channels

- Process P_i = *program graph* *PG*$_i$ + *communication actions*

$c!e$	transmit the value of expression e along channel c
$c?x$	receive a message via channel c and assign it to variable x

- $\mathit{Comm} \ = \ \{ \ c!e, \ c?x \ \mid \ c \in \mathit{Chan}, \ e \in \mathit{Expr}, \ x \in \mathit{Var}. \ \ \mathit{dom}(x) \supseteq \mathit{dom}(c) = \mathit{dom}(e) \ \}$

- Sending and receiving a message
  - $c!e$ puts the value of e at the rear of the buffer c (if c is not full)
  - $c?x$ retrieves the front element of the buffer and assigns it to x (if c is not empty)
  - if $\mathit{cap}(c) = 0$, channel c has *no* buffer
  - if $\mathit{cap}(c) = 0$, sending and receiving can takes place simultaneously
    this is called *synchronous message passing* or *handshaking*
  - if $\mathit{cap}(c) > 0$, sending and receiving can never take place simultaneously
    this is called *asynchronous message passing*

Channel systems

A program graph over $(\mathit{Var}, \mathit{Chan})$ is a tuple

$$\mathit{PG} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \rightarrow, \mathit{Loc}_0, g_0)$$

where

$$\rightarrow \quad \subseteq \quad \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times (\mathit{Act} \cup \mathit{Comm}) \times \mathit{Loc}$$

A *channel system* CS over $(\bigcup_{0<i\leqslant n} \mathit{Var}_i, \mathit{Chan})$:

$$\mathit{CS} \;=\; [\mathit{PG}_1 \mid \ldots \mid \mathit{PG}_n]$$

with program graphs PG_i over $(\mathit{Var}_i, \mathit{Chan})$

Communication actions

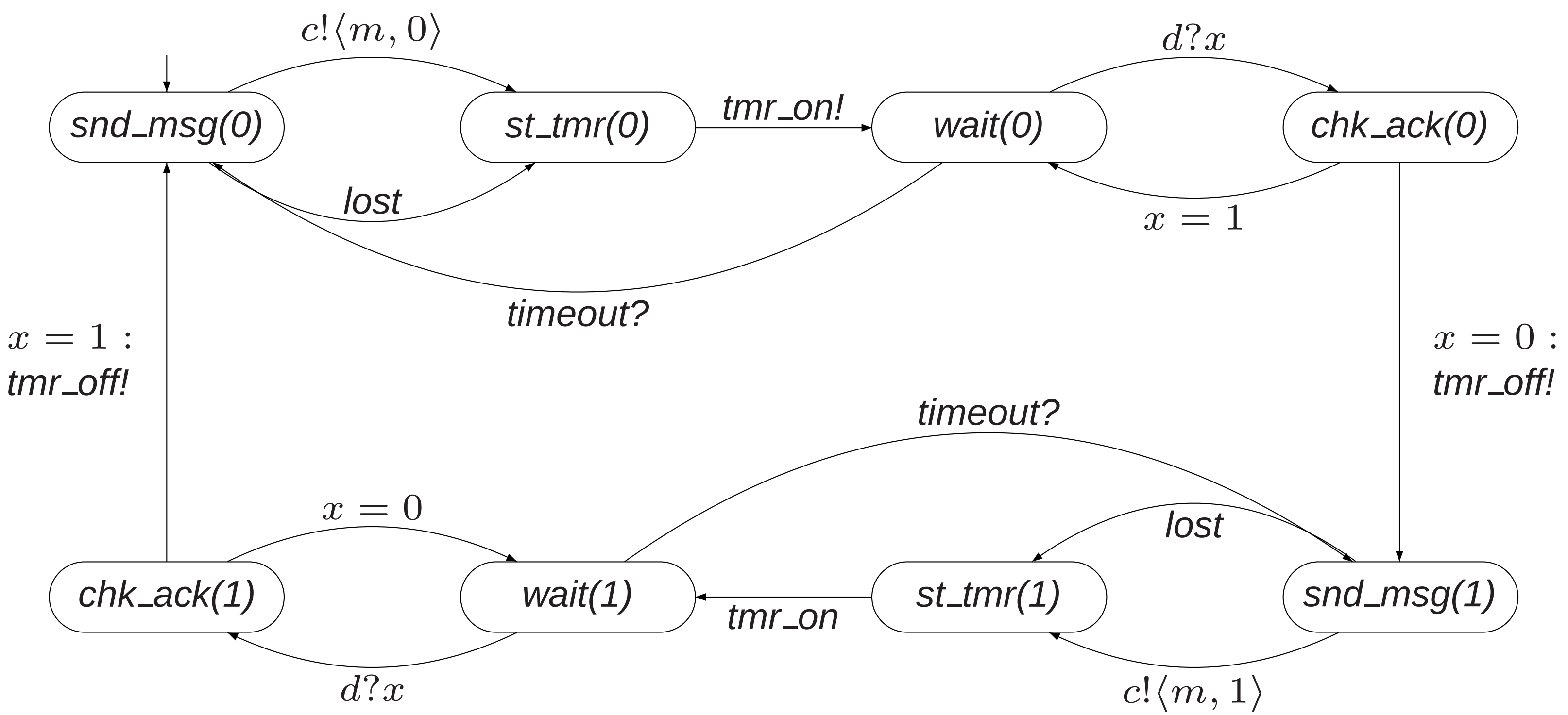
- *Handshaking*
  - if *cap*$(c) = 0$, then process P_i can perform $\ell_i \xrightarrow{c!e} \ell_i'$ only
  - ... if P_j, say, can perform $\ell_j \xrightarrow{c?x} \ell_j'$
  - the effect corresponds to the (atomic) *distributed* assignment $x :=$ *value*(e)

- *Asynchronous message passing*
  - if *cap*$(c) > 0$, then process P_i can perform $\ell_i \xrightarrow{c!e} \ell_i'$
  - ... if and only if less than *cap*(c) messages are stored in c
  - P_j may perform $\ell_j \xrightarrow{c?x} \ell_j'$ if and only if the buffer of c is not empty
  - then the first element of the buffer is extracted and assigned to x (atomically)

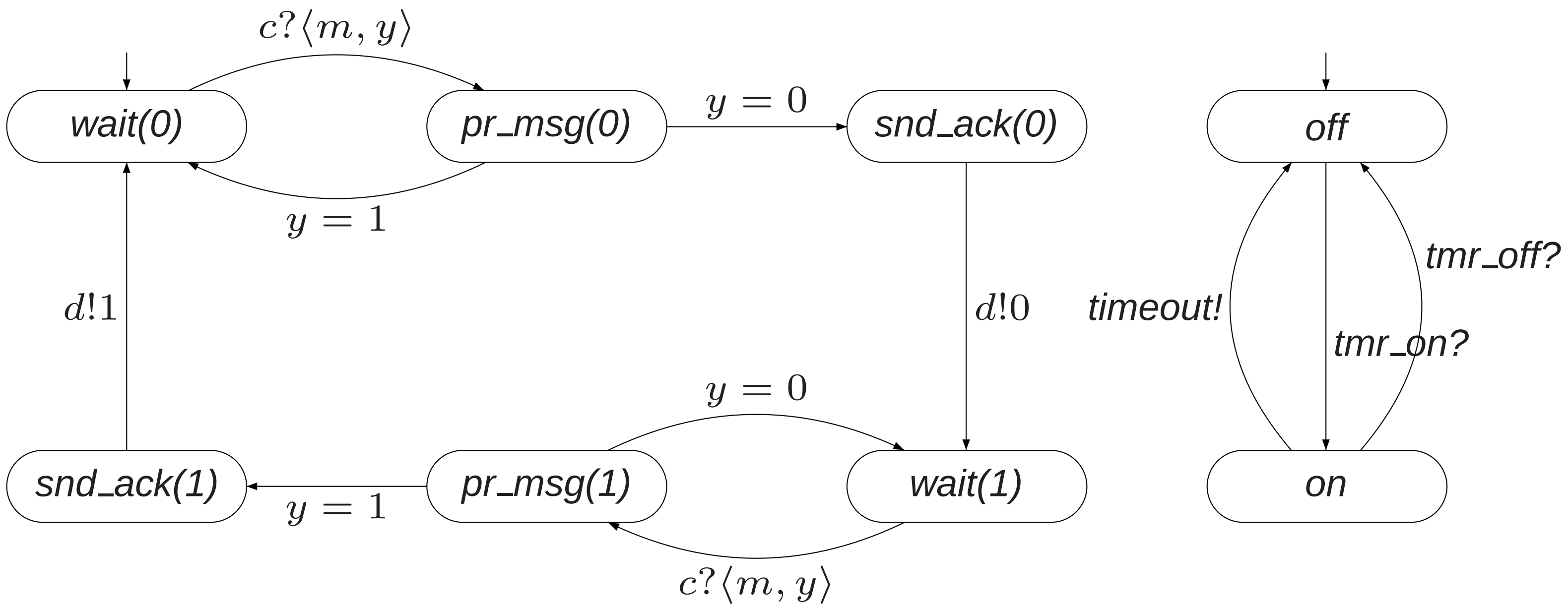
	executable if ...	effect
$c!e$	c is not "full"	$Enqueue(c, value(e))$
$c?x$	c is not empty	$\langle x := Front(c) \,;\, Dequeue(c) \rangle;$

The alternating bit protocol

The alternating bit protocol: sender



The alternating bit protocol: receiver



Channel evaluations

- A *channel evaluation* ξ is
  - a mapping from channel $c \in$ *Chan* onto a sequence $\xi(c) \in$ *dom*$(c)^*$ such that
  - current length cannot exceed the capacity of c: *len*$(\xi(c)) \leqslant$ *cap*(c)
  - $\xi(c) = v_1\, v_2\, \ldots\, v_k$ (*cap*$(c) \geqslant k$) denotes v_1 is at front of buffer etc.

- $\xi[c := v_1 \ldots v_k]$ denotes the channel evaluation

$$\xi[c := v_1 \ldots v_k](c') \; = \; \begin{cases} \xi(c') & \text{if } c \neq c' \\ v_1 \ldots v_k & \text{if } c = c'. \end{cases}$$

- Initial channel evaluation ξ_0 equals $\xi_0(c) = \varepsilon$ for any c

Transition system semantics of a channel system

Let *CS* $= [PG_1 \mid \ldots \mid PG_n]$ be a *channel system* over $(\mathit{Chan}, \mathit{Var})$ with

$$PG_i = (\mathit{Loc}_i, \mathit{Act}_i, \mathit{Effect}_i, \rightsquigarrow_i, \mathit{Loc}_{0,i}, g_{0,i}), \qquad \text{for } 0 < i \leqslant n$$

$\mathit{TS}(\mathit{CS})$ is the *transition system* $(S, \mathit{Act}, \rightarrow, I, \mathit{AP}, L)$ where:

- $S \;=\; (\mathit{Loc}_1 \times \ldots \times \mathit{Loc}_n) \times \mathit{Eval}(\mathit{Var}) \times \mathit{Eval}(\mathit{Chan})$
- $\mathit{Act} \;=\; \left(\biguplus_{0<i\leqslant n} \mathit{Act}_i\right) \uplus \{\,\tau\,\}$
- $\rightarrow$ is defined by the inference rules on the next slides
- $I = \left\{ \langle \ell_1, \ldots, \ell_n, \eta, \xi_0 \rangle \;\mid\; \forall i.\ (\ell_i \in Loc_{0,i} \wedge \eta \models g_{0,i}) \wedge \forall c.\ \xi_0(c) = \varepsilon \right\}$
- $\mathit{AP} \;=\; \biguplus_{0<i\leqslant n} \mathit{Loc}_i \;\uplus\; \mathit{Cond}(\mathit{Var})$
- $L(\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle) \;=\; \{\, \ell_1, \ldots, \ell_n \,\} \cup \{\, g \in \mathit{Cond}(\mathit{Var}) \mid \eta \models g \,\}$

Inference rules (I)

- Interleaving for $\alpha \in \mathit{Act}_i$:

$$\frac{\ell_i \xrightarrow{g:\alpha} \ell_i' \quad \wedge \quad \eta \models g}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\alpha} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi \rangle}$$

where $\eta' = \mathit{Effect}(\alpha, \eta)$

- Synchronous message passing over $c \in \mathit{Chan}$, $\mathit{cap}(c) = 0$:

$$\frac{\ell_i \xrightarrow{g:c?x} \ell_i' \ \wedge \ \ell_j \xrightarrow{g':c!e} \ell_j' \ \wedge \ \eta \models g \wedge g' \quad \wedge \quad i \neq j}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_j', \ldots, \ell_n, \eta', \xi \rangle}$$

where $\eta' = \eta[x := \eta(e)]$.

Inference rules (II)

- Asynchronous message passing for $c \in \mathit{Chan}$, $\mathit{cap}(c) > 0$:
  - receive a value along channel c and assign it to variable x:

$$\frac{\ell_i \xrightarrow{g:c?x} \ell_i' \ \wedge\ \eta \models g \ \wedge\ \mathit{len}(\xi(c)) = k > 0 \ \wedge\ \xi(c) \ = \ v_1 \ldots v_k}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi' \rangle}$$

where $\eta' = \eta[x := v_1]$ and $\xi' = \xi[c := v_2 \ldots v_k]$.

  - transmit value $\eta(e) \in \mathit{dom}(c)$ over channel c:

$$\frac{\ell_i \xrightarrow{g:c!e} \ell_i' \ \wedge\ \eta \models g \ \wedge\ \mathit{len}(\xi(c)) = k < \mathit{cap}(c) \ \wedge\ \xi(c) \ = \ v_1 \ldots v_k}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta, \xi' \rangle}$$

where $\xi' = \xi[c := v_1\, v_2 \ldots v_k\, \eta(e)]$.

Handling unexpected messages

sender S	timer	receiver R	channel c	channel d	event
$snd\_msg(0)$	off	$wait(0)$	$\varnothing$	$\varnothing$	
$st\_tmr(0)$	off	$wait(0)$	$\langle m, 0\rangle$	$\varnothing$	message with bit 0 transmitted
$wait(0)$	on	$wait(0)$	$\langle m, 0\rangle$	$\varnothing$	
$snd\_msg(0)$	off	$wait(0)$	$\langle m, 0\rangle$	$\varnothing$	timeout
$st\_tmr(0)$	off	$wait(0)$	$\langle m, 0\rangle\ \langle m, 0\rangle$	$\varnothing$	retransmission
$st\_tmr(0)$	off	$pr\_msg(0)$	$\langle m, 0\rangle$	$\varnothing$	receiver reads first message
$st\_tmr(0)$	off	*snd_ack*(0)	$\langle m, 0\rangle$	$\varnothing$	
$st\_tmr(0)$	off	$wait(1)$	$\langle m, 0\rangle$	0	receiver changes into mode-1
$st\_tmr(0)$	off	$pr\_msg(1)$	$\varnothing$	0	receiver reads retransmission
$st\_tmr(0)$	off	$wait(1)$	$\varnothing$	0	and ignores it
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

nanoPromela

- Promela (Process Meta Language) is modeling language for SPIN
  - SPIN = most widely used model checker
  - developed by Gerard Holzmann (Bell Labs, NASA JPL)
  - ACM Software Award 2002
- nanoPromela is the core of Promela
  - shared variables and channel-based communication
  - formal semantics of a Promela model is a channel system
  - processes are defined by means of a guarded command language
- No actions, statements describe effect of actions

nanoPromela

nanoPromela-program $\overline{\mathcal{P}} = [\mathcal{P}_1 | \ldots | \mathcal{P}_n]$ with $\mathcal{P}_i$ a process

A process is specified by a statement:

$$
\begin{array}{lll}
\text{stmt} & ::= & \texttt{skip} \mid x := \text{expr} \mid c?x \mid c!\text{expr} \mid \\
 & & \text{stmt}_1 \,;\, \text{stmt}_2 \mid \texttt{atomic}\{\text{assignments}\} \mid \\
 & & \textbf{if} \quad :: g_1 \Rightarrow \text{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \text{stmt}_n \quad \textbf{fi} \quad \mid \\
 & & \textbf{do} \quad :: g_1 \Rightarrow \text{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \text{stmt}_n \quad \textbf{do} \\
\text{assignments} & ::= & x_1 := \text{expr}_1 \,;\, x_2 := \text{expr}_2 \,;\, \ldots \,;\, x_m := \text{expr}_m
\end{array}
$$

x is a variable in *Var*, expr an expression and c a channel, g_i a guard

assume the Promela specification is type-consistent

Conditional statements

$$\textbf{if} \ :: g_1 \Rightarrow \text{stmt}_1 \ \ldots \ :: g_n \Rightarrow \text{stmt}_n \ \textbf{fi}$$

- Nondeterministic choice between statements stmt_i for which g_i holds
- Test-and-set semantics: (deviation from Promela)
  - guard evaluation + selection of enabled command + execution first atomic step of selected statement is all performed atomically
- The **if**–**fi**–command blocks if no guard holds
  - parallel processes may unblock a process by changing shared variables
  - e.g., when $y{=}0$, **if** $:: y > 0 \Rightarrow x := 42$ **fi** waits until y exceeds 0
- Standard abbreviations:
  - **if** g **then** stmt_1 **else** stmt_2 **fi** $\equiv$ **if** $:: g \Rightarrow \text{stmt}_1 \ :: \neg g \Rightarrow \text{stmt}_2$ **fi**
  - **if** g **then** stmt_1 **fi** $\equiv$ **if** $:: g \Rightarrow \text{stmt}_1 \ :: \neg g \Rightarrow$ `skip` **fi**

Iteration statements

$$\textbf{do} \;\; :: g_1 \Rightarrow \text{stmt}_1 \; \ldots \; :: g_n \Rightarrow \text{stmt}_n \; \textbf{od}$$

- Iterative execution of nondeterministic choice among $g_i \Rightarrow \text{stmt}_i$
  - where guard g_i holds in the current state
- No blocking if all guards are violated; instead, loop is aborted
- **do** $:: g \Rightarrow$ stmt **od** $\equiv$ **while** g **do** stmt **od**
- No break-statements to abort a loop (deviation from Promela)

Peterson's algorithm

The nanoPromela-code of process $\mathcal{P}_1$ is given by the statement:

$$
\begin{array}{lll}
\textbf{do} & :: \; \text{true} \Rightarrow & \texttt{skip}; \\
& & \texttt{atomic}\{b_1 := \text{true}; x := 2\}; \\
& & \textbf{if} \;\; :: \; (x = 1) \vee \neg b_2 \;\Rightarrow\; \mathit{crit}_1 := \text{true} \;\; \textbf{fi} \\
& & \texttt{atomic}\{\mathit{crit}_1 := \text{false}; b_1 := \text{false}\} \\
\textbf{od} & &
\end{array}
$$

Beverage vending machine

The following nanoPromela program describes its behaviour:

$$
\begin{array}{lll}
\textbf{do} & :: & \text{true} \Rightarrow \\
& & \quad \texttt{skip}; \\
& & \quad \textbf{if} \quad :: \; \mathit{nsprite} > 0 \;\Rightarrow\; \mathit{nsprite} := \mathit{nsprite} - 1 \\
& & \quad \phantom{\textbf{if}} \quad :: \; \mathit{nbeer} > 0 \;\Rightarrow\; \mathit{nbeer} := \mathit{nbeer} - 1 \\
& & \quad \phantom{\textbf{if}} \quad :: \; \mathit{nsprite} = \mathit{nbeer} = 0 \Rightarrow \texttt{skip} \\
& & \quad \textbf{fi} \\
& :: & \text{true} \Rightarrow \texttt{atomic}\{\mathit{nbeer} := \mathit{max}; \mathit{nsprite} := \mathit{max}\} \\
\textbf{od} & &
\end{array}
$$

Formal semantics

The *semantics* of a nanoPromela-statement over $(Var, Chan)$ is a *program graph* over $(Var, Chan)$.

The program graphs $PG_1, \ldots, PG_n$ for the processes $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of a nanoPromela-program $\overline{\mathcal{P}} = [\mathcal{P}_1 | \ldots | \mathcal{P}_n]$ constitute a *channel system* over $(Var, Chan)$

Example:

$$\begin{array}{lllllll} \text{loop} & = & \textbf{do} & :: & x > 1 & \Rightarrow & y := x + y \\ & & & :: & y < x & \Rightarrow & x := 0;\ y := x \\ & & \textbf{od} & & & & \end{array}$$

Sub-statements

Inference rules

$$\frac{}{\texttt{skip} \xrightarrow{\text{true: } \textit{id}} \text{exit}}$$

where *id* denotes an action that does not change the values of the variables

$$\frac{}{x := \text{expr} \xrightarrow{\text{true} \,:\, \text{assign}(x, \text{expr})} \text{exit}}$$

$\text{assign}(x, \text{expr})$ denotes the action that only changes x, no other variables

$$\frac{}{c?x \xrightarrow{\text{true} \,:\, c?x} \text{exit}} \qquad \frac{}{c!\text{expr} \xrightarrow{\text{true} \,:\, c!\text{expr}} \text{exit}}$$

Inference rules

$$\frac{}{\texttt{atomic}\{x_1 := \mathsf{expr}_1; \ldots ; x_m := \mathsf{expr}_m\} \xrightarrow{\mathsf{true}\;:\;\alpha_m} \mathsf{exit}}$$

where $\alpha_0 = \mathit{id}$, $\alpha_i = \mathit{Effect}(\mathsf{assign}(x_i, \mathsf{expr}_i), \mathit{Effect}(\alpha_{i-1}, \eta))$ for $1 \leqslant i \leqslant m$

$$\frac{\mathsf{stmt}_1 \xrightarrow{g:\alpha} \mathsf{stmt}_1' \neq \mathsf{exit}}{\mathsf{stmt}_1 ; \mathsf{stmt}_2 \xrightarrow{g:\alpha} \mathsf{stmt}_1' ; \mathsf{stmt}_2}$$

$$\frac{\mathsf{stmt}_1 \xrightarrow{g:\alpha} \mathsf{exit}}{\mathsf{stmt}_1 ; \mathsf{stmt}_2 \xrightarrow{g:\alpha} \mathsf{stmt}_2}$$

Inference rules

$$\frac{\mathsf{stmt}_i \xrightarrow{h:\alpha} \mathsf{stmt}'_i}{\mathsf{cond\_cmd} \xrightarrow{g_i \wedge h:\alpha} \mathsf{stmt}'_i}$$

$$\frac{\mathsf{stmt}_i \xrightarrow{h:\alpha} \mathsf{stmt}'_i \neq \mathsf{exit}}{\mathsf{loop} \xrightarrow{g_i \wedge h:\alpha} \mathsf{stmt}'_i; \mathsf{loop}} \qquad \frac{\mathsf{stmt}_i \xrightarrow{h:\alpha} \mathsf{exit}}{\mathsf{loop} \xrightarrow{g_i \wedge h:\alpha} \mathsf{loop}}$$

$$\frac{}{\mathsf{loop} \xrightarrow{\neg g_1 \wedge \ldots \wedge \neg g_n} \mathsf{exit}}$$