Real-Time Specification and Refinement

Ian J. Hayes

School of Information Technology and Electrical Engineering,

The University of Queensland, Brisbane, 4072, Australia

August 29, 2010

The Presentation

$\{\tau \leq \mathit{Start\_Time}\}$

Present Overview;

do $\neg$ *Complete* $\rightarrow$

Present Slide;

Answer Questions;

deadline *Finish_Time* $- 10min$

od;

deadline *Finish_Time* $- 10min$;

Answer Questions;

deadline *Finish_Time*

Overview

- Specifying real-time systems
  - Current time variable τ
  - Input and output traces
  - Local and auxiliary variables
- Stepwise refinement
  - Idle-stable guards
  - Idle-invariant properties
- Machine-independent real-time programs
  - Real-time behaviour of dependent on compiler/machine/...
  - The deadline command
- Timing constraint analysis

Time

A simple way to model time is to add a special time variable, τ, representing the "current" time.
Time does not go backwards.

$$\tau_0 \leq \tau$$

where

- τ_0 represents the start time of a command, and
- τ its finish time.

Commands may execute forever, therefore we allow τ to take on the value infinity (∞).

Machine independence and real time

- The most significant advantage of programming languages over assembly languages is machine independence.
- However, the execution time of (paths through) a program depends on the particular implementation.
- Hence the real-time characteristics of these languages are machine dependent.

The deadline command

To overcome this problem we introduce

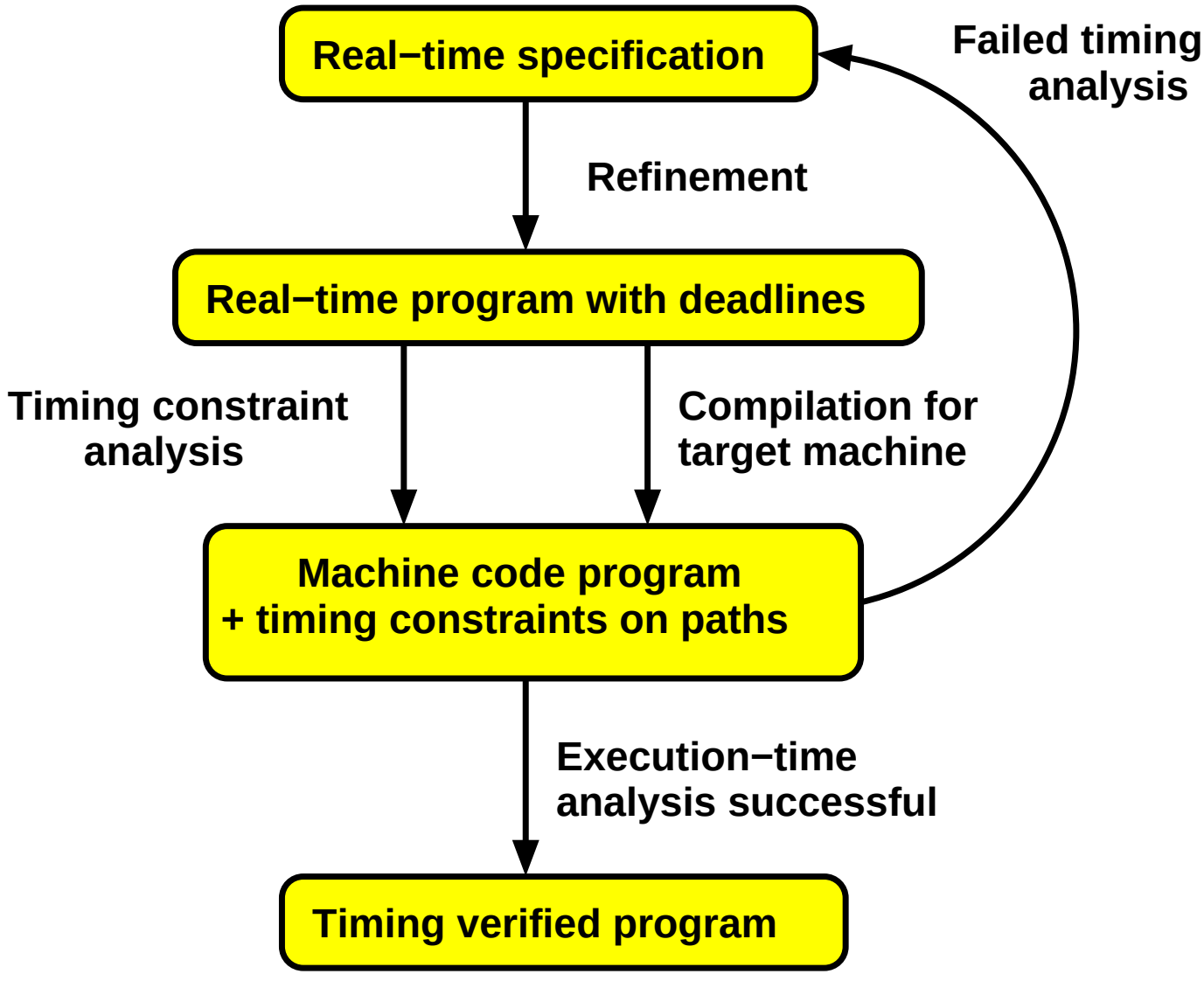
- a (theoretical) deadline command, **deadline** *D*
- it guarantees to finish by absolute time *D*
- takes no time to execute
- it has no possible behaviour if it is executed at a time strictly after its specified deadline (it is magic) and in this case it cannot be possibly be implemented
- for example

$\{\tau \leq s\}$ -- assume time is before s

$C;$ -- command that does not modify s

deadline $s + 1\mathrm{ms}$

Development process



Defining the deadline command

Within the postcondition, τ stands for the finish time of the command and τ_0 for the start time.

$$\textbf{deadline}\ D \ \widehat{=}\ \left[\tau_0 = \tau \wedge \tau \leq D\right]$$

- If a deadline is "executed" at or before D it is a no-op
- If a deadline is "executed" strictly after D, then the deadline command has no possible behaviour (it is magic) and cannot be implemented
- To ensure a deadline can always be implemented, we need to carry out a timing analysis to ensure that all deadlines will be reached before they expire

Get the current time

- t : **gettime** assigns t an approximation to the current time
- An implementation cannot accurately represent all real numbers (as in type *Time*) and hence the type of t will be some subset of *Time*.
- A simple definition of the **gettime** command is

$$t : \textbf{gettime} \;\widehat{=}\; t : \left[\tau_0 \leq t \leq \tau\right]$$

Read from an external input

- We assume the value of an external input is under the control of the environment and not the program.
- For real-time programs we would like to consider not only discrete inputs but also analog inputs (sampled by an analog-to-digital converter). Hence we model an external input (or output) as a function from *Time* to the type of the input.
- A simple definition of a read command assumes that the value read is the value of the input at some time during the execution of the read command:

$$x : \mathbf{read}(i) \;\widehat{=}\; x : \left[\exists t : [\tau_0 \ldots \tau] \bullet x = i(t)\right]$$

Variables

We refer to the set of variables in scope as the environment, and use the name ρ for the environment.
In real-time programs we distinguish four kinds of variables:

- inputs, $\rho.in$, which are under external control;
- outputs, $\rho.out$, which are under the control of the program;
- local variables, $\rho.local$, which are under the control of the program, but unlike outputs are not externally visible; and
- auxiliary variables, $\rho.aux$, which are similar to local variables, but are restricted to appear only in assumptions, specifications, deadline commands and assignments to auxiliary variables.

Inputs and outputs

- Inputs and outputs are modelled as functions from *Time* to the declared type of the variable, e.g.,

  input *sensor* : *boolean*;

  output *barrier* : $\{up, down\}$;

- *sensor* is modelled as a function from *Time* to boolean, and
- *barrier* is modelled as a function from *Time* to $\{up, down\}$.
- The expression $sensor(t)$ gives the value of *sensor* at time t.
- Note that it is not meaningful to talk about the value of a variable at time infinity.

Null commands

- **skip** does nothing and takes no time

$$\textbf{skip} \;\widehat{=}\; \left[\tau_0 = \tau\right]$$

- **idle** does nothing but may take time but does terminate

$$\textbf{idle} \;\widehat{=}\; \left[\mathit{true}\right]$$

- **bone_idle** does nothing but may take time and may not terminate

$$\textbf{bone\_idle} \;\widehat{=}\; \infty\left[\mathit{true}\right]$$

From lazy to the bone.

Defining the delay command

- The delay command waits until absolute time D, as evaluated at the start time of the delay command, τ_0.
- We use the notation $D \mathbin{@} \tau_0$ to stand for the value of the expression D at time τ_0

$$\textbf{delay until } D \quad \widehat{=} \quad \left[(D \mathbin{@} \tau_0) \leq \tau \right]$$

- What does the following mean?

delay until $\tau + 1$

- If i is an external input, what does the following mean?

delay until i

Idle-stable expressions

- As the delay command may take time to execute, we restrict the expressions allowed for the delay time D to idle-stable expressions: expressions that do not change their value while being evaluated.
- The main problem is with expressions that refer to
  - the current time or
  - to the value of external inputs (whose values may change independently over time).
- An expression, D, is idle-stable if it is stable over the execution of the **idle** command: i.e., provided for all X

$$\left[D = X \wedge \tau = \tau_0\right] ; \textbf{idle} \quad = \quad \textbf{idle}; \left[D = X \wedge \tau = \tau_0\right]$$

Idle-stable expressions

D is idle-stable provided for all times τ_0 and τ,

$$\tau_0 \leq \tau < \infty \wedge \mathit{stable}(\mathit{outputs}, [\tau_0 \ldots \tau]) \Rrightarrow D \mathbin{@} \tau_0 = D \mathbin{@} \tau$$

where

- *outputs* is the set of all the externally visible outputs of the program
- $\mathit{stable}(\mathit{outputs}, [\tau_0 \ldots \tau])$ means that these outputs do not change over the closed time interval from τ_0 to τ
- references to local variables in the expressions $D \mathbin{@} \tau_0$ and $D \mathbin{@} \tau$ refer to the same variable to represent the fact that local variables are unchanged over an idle period.

Stable

- We define the predicate *stable* by

$$stable(v, S) \mathrel{\widehat{=}} S \neq \{\} \Rightarrow (\exists x \bullet v(\!| S |\!) = \{x\})$$

  where $v(\!| S |\!)$ is the image of the set S through the function v.
- We allow the first argument of stable to be a vector of variables, in which case all variables in the vector are stable.
- To specify the closed interval of times from s until t, we use the notation $[s \mathinner{..} t]$.
- The open interval is specified by $(s \mathinner{..} t)$.
- We also allow half-open, half-closed intervals.

Exercises

D is idle-stable provided for all times τ_0 and τ,

$$\tau_0 \leq \tau < \infty \wedge \mathit{stable}(\mathit{outputs}, [\tau_0 \ldots \tau]) \Rrightarrow D \mathbin{@} \tau_0 = D \mathbin{@} \tau$$

Which of the following expressions are idle-stable?

- 1
- $z(\tau)$, where z is an output
- τ
- v, where v is a local variable
- $i(\tau)$

Exercises

D is idle-stable provided for all times τ_0 and τ,

$$\tau_0 \leq \tau < \infty \wedge \mathit{stable}(\mathit{outputs}, [\tau_0 \ldots \tau]) \Rrightarrow D \mathbin{@} \tau_0 = D \mathbin{@} \tau$$

Which of the following expressions are idle-stable?

- 1

  Yes $\quad \tau_0 \leq \tau < \infty \wedge \mathit{stable}(o, [\tau_0 \ldots \tau]) \Rrightarrow 1 = 1$

- $z(\tau)$, where z is an output

  Yes $\quad \tau_0 \leq \tau < \infty \wedge \mathit{stable}(z, [\tau_0 \ldots \tau]) \Rrightarrow z(\tau_0) = z(\tau)$

Exercises...

Which of the following expressions are idle-stable?

- τ

  No $\quad \tau_0 \leq \tau < \infty \wedge \mathit{stable}(o, [\tau_0 \ldots \tau]) \not\Rightarrow \tau_0 = \tau$

- v, where v is a local variable

  Yes $\quad \tau_0 \leq \tau < \infty \wedge \mathit{stable}(o, [\tau_0 \ldots \tau]) \Rightarrow v = v$

- $i(\tau)$

  No $\quad \tau_0 \leq \tau < \infty \wedge \mathit{stable}(o, [\tau_0 \ldots \tau]) \not\Rightarrow i(\tau_0) = i(\tau)$

If an expression has no references to external inputs or τ then it is idle-stable.

Refinement calculus

- Our approach is based on the refinement calculus of Back and Morgan.
- The programming language is extended with a specification command to give a wide-spectrum language.
- This allows both specifications and code to be treated within the same framework.
- In particular, it allows components of programs to be (as yet to be developed into code) specifications.

Semantics

- We represent the semantics of a command by a predicate in a form similar to that of Hehner [Heh89, Heh93], and Hoare and He [HH98].
- The predicate relates the initial (zero-subscripted) and final (unsubscripted) values of the state variables as well as constraining the traces of the outputs over time.
- All our commands insist that time does not go backwards:

$$\tau_0 \leq \tau$$

Refinement

- The meaning function, $\mathcal{M}$, takes the variables in scope, ρ, and a command C and returns the corresponding predicate, $\mathcal{M}_\rho(C)$.
- Refinement of commands (in an environment, ρ) is defined as reverse entailment:

$$C \sqsubseteq_\rho D \mathrel{\widehat{=}} \mathcal{M}_\rho(C) \Leftarrow \mathcal{M}_\rho(D)$$

where '$P \Leftarrow Q$' holds if for all possible values of the variables, whenever Q holds, P holds.

Real-time specification command

We introduce a possibly nonterminating real-time specification command,

$$\infty\vec{x}: \left[Q \right]$$

where

- $\vec{x}$ is a vector of variables called the frame
- The relation Q is its postcondition (or effect) expressed as a predicate between a zero-subscripted pre state and an unsubscripted post state
- The '∞' at the beginning is just part of the syntax; it reminds us that the command might not terminate.

Terminating specification command

The only difference is the extra requirement that the effect should achieve $\tau < \infty$.

$$\vec{x}\colon \left[Q \right] \mathrel{\widehat{=}} \infty\vec{x}\colon \left[Q \wedge \tau < \infty \right]$$

For example, in the following specification *before* is an auxiliary time-valued variable representing the start time of the command. The task to be completed is to sample *sensor* between *before* and $event + lim$, and terminate before $event + lim$.

$$\left\{ before = \tau \right\} sens\colon \left[\begin{array}{c} sens \in sensor\left(\!\left| \, [before \ldots event + lim] \, \right|\!\right) \wedge \\ \tau \leq event + lim \end{array} \right]$$

Specification frame

- The frame, $\vec{x}$, of a specification command lists those variables that may be modified by the command.
- The frame may not include inputs.
- The current time variable, τ, is implicitly in the frame.
- All outputs not in the frame, i.e., those in $\rho.out$ but not $\vec{x}$, are defined to be stable for the duration of the command, provided the assumption holds initially.

Weaken precondition

Law 1 (weaken precondition) *Provided* $P_1 \Rrightarrow P_2$,

$$\left\{P_1\right\} \infty\vec{x}\colon \left[Q\right] \sqsubseteq \left\{P_2\right\} \infty\vec{x}\colon \left[Q\right]$$

For example,

$$\left\{before = \tau\right\}\ sens\colon \left[\begin{array}{c} sens \in sensor(\!|\ [\tau_0 \ldots \tau]\ |\!) \wedge \\ \tau \leq event + lim \end{array}\right]$$

$\sqsubseteq$ Law 1 (weaken precondition)

$$\left\{true\right\}\ sens\colon \left[\begin{array}{c} sens \in sensor(\!|\ [\tau_0 \ldots \tau]\ |\!) \wedge \\ \tau \leq event + lim \end{array}\right]$$

$$\sqsubseteq\ sens\colon \left[\begin{array}{c} sens \in sensor(\!|\ [\tau_0 \ldots \tau]\ |\!) \wedge \\ \tau \leq event + lim \end{array}\right]$$

Strengthen postcondition

One can take into account the following:

- time cannot go backwards: $\tau_0 \leq \tau$
- the command starts at a finite time: $\tau_0 < \infty$
- the precondition holds for the initial state of the variables: $P \mathbin{@} \tau_0$, where $P \mathbin{@} \tau_0$ is P with τ and all local and auxiliary variables in the frame replaced by their zero-subscripted variants
- outputs and local and auxiliary variables not in the frame are stable

$$stable(\rho.out \setminus \vec{x}, [\tau_0 \mathinner{..} \tau])$$
$$\forall v : (\rho.local \cup \rho.aux) \setminus \vec{x} \bullet v_0 = v$$

Strengthen postcondition law

Law 2 (strengthen postcondition) *Provided*

$$\left(\begin{array}{l} \tau_0 \leq \tau \wedge \tau_0 < \infty \wedge P \mathbin{@} \tau_0 \wedge \\ \mathit{stable}(\rho.\mathit{out} \setminus \vec{x}, [\tau_0 \mathinner{..} \tau]) \wedge \\ (\forall v : (\rho.\mathit{local} \cup \rho.\mathit{aux}) \setminus \vec{x} \bullet v_0 = v) \wedge \\ R \end{array}\right) \Rrightarrow Q$$

then

$$\left\{P\right\} \infty\vec{x}\colon \left[Q\right] \sqsubseteq \left\{P\right\} \infty\vec{x}\colon \left[R\right]$$

Example

$$\left\{ \mathit{before} = \tau \right\} \mathit{sens}: \left[\begin{array}{c} \mathit{sens} \in \mathit{sensor}\left(\!\left| \, [\mathit{before} \mathinner{..} \mathit{event} + \mathit{lim}] \, \right|\!\right) \wedge \\ \tau \leq \mathit{event} + \mathit{lim} \end{array} \right]$$

$\sqsubseteq$ Law 2 (strengthen postcondition)

$$\left\{ \mathit{before} = \tau \right\} \mathit{sens}: \left[\begin{array}{c} \mathit{sens} \in \mathit{sensor}\left(\!\left| \, [\tau_0 \mathinner{..} \tau] \, \right|\!\right) \wedge \\ \tau \leq \mathit{event} + \mathit{lim} \end{array} \right]$$

provided

$$\begin{array}{l} \tau_0 \leq \tau \wedge \tau_0 < \infty \wedge \mathit{before} = \tau_0 \wedge \\ \mathit{sens} \in \mathit{sensor}\left(\!\left| \, [\tau_0 \mathinner{..} \tau] \, \right|\!\right) \wedge \tau \leq \mathit{event} + \mathit{lim} \wedge \\ \mathit{stable}\left(\mathit{barrier}, [\tau_0 \mathinner{..} \tau]\right) \wedge \mathit{before} = \mathit{before}_0 \end{array}$$

$$\Rrightarrow \mathit{sens} \in \mathit{sensor}\left(\!\left| \, [\mathit{before} \mathinner{..} \mathit{event} + \mathit{lim}] \, \right|\!\right) \wedge \tau \leq \mathit{event} + \mathit{lim}$$

An invalid law

The following standard law refines a specification command to a conditional with a specification command in each branch.

$$\left\{P\right\}\, x{:}\left[Q\right] \not\sqsubseteq \textbf{if } B \textbf{ then } \left\{P \wedge B\right\}\, x{:}\left[Q\right] \textbf{ else } \left\{P \wedge \neg B\right\}\, x{:}\left[Q\right]$$

Consider the following example

$$\left\{\tau \leq D\right\}\, x{:}\left[R \wedge \tau - \tau_0 \leq 3\right]$$
$$\not\sqsubseteq \textbf{if } B \textbf{ then } \left\{B \wedge \tau \leq D\right\}\, x{:}\left[R \wedge \tau - \tau_0 \leq 3\right]$$
$$\textbf{else } \left\{\neg B \wedge \tau \leq D\right\}\, x{:}\left[R \wedge \tau - \tau_0 \leq 3\right]$$

Idle invariance

P is idle invariant provided if P holds at a time τ_0 then it holds at any later time τ, provided none of the variables under the control of the program change.

$$\left\{P\right\}\ \textbf{idle}\ \sqsubseteq\ \left\{P\right\}\ \textbf{idle}\ \left\{P\right\}$$

A predicate P is idle-invariant if for all τ_0 and τ,

$$\tau_0 \leq \tau < \infty \wedge \mathit{stable}(\mathit{outputs}, [\tau_0 \ldots \tau]) \wedge P\ @\ \tau_0 \Rrightarrow P\ @\ \tau$$

Which of the following predicates are idle-invariant?

- $\tau \leq D$
- $D \leq \tau$
- $i(\tau) = X$

Exercises...

- $\tau \leq D$ is not idle-invariant

  $\tau_0 \leq \tau < \infty \wedge \mathit{stable}(o, [\tau_0 \ldots \tau]) \wedge \tau_0 \leq D \not\Rrightarrow \tau \leq D$

- $D \leq \tau$ is idle-invariant

  $\tau_0 \leq \tau < \infty \wedge \mathit{stable}(o, [\tau_0 \ldots \tau]) \wedge D \leq \tau_0 \Rrightarrow D \leq \tau$

- $i(\tau) = X$ is not idle-invariant

  $\tau_0 \leq \tau < \infty \wedge \mathit{stable}(o, [\tau_0 \ldots \tau]) \wedge i(\tau_0) = X \not\Rrightarrow i(\tau) = X$

If P does not refer to τ (or to inputs at τ) then it is idle-invariant

Pre-idle-invariant

- A relation Q is pre-idle-invariant provided

$$x{:}\left[Q\right] \quad \sqsubseteq \quad \textbf{idle};\ x{:}\left[Q\right]$$

- Q is pre-idle-invariant provided for all u, τ_0, and τ,

$$u \leq \tau_0 \leq \tau \wedge \tau_0 < \infty \wedge \mathit{stable}(\mathit{outputs}, [u \mathinner{..} \tau_0]) \wedge Q \Rrightarrow Q\,[u/\tau_0]$$

Which of the following predicates is pre-idle-invariant?

- $\tau \leq D$
- $\tau_0 \leq D$
- $D \leq \tau_0$
- $\tau - \tau_0 \leq 3$

Pre-idle-invariant exercises...

- $\tau \leq D$ is pre-idle-invariant
  $u \leq \tau_0 \leq \tau \wedge \tau_0 < \infty \wedge \mathit{stable}(o, [u \ldots \tau_0]) \wedge \tau \leq D \Rrightarrow \tau \leq D$
- $\tau_0 \leq D$ is pre-idle-invariant
  $u \leq \tau_0 \leq \tau \wedge \tau_0 < \infty \wedge \mathit{stable}(o, [u \ldots \tau_0]) \wedge \tau_0 \leq D \Rrightarrow u \leq D$
- $D \leq \tau_0$ is not pre-idle-invariant
  $u \leq \tau_0 \leq \tau \wedge \tau_0 < \infty \wedge \mathit{stable}(o, [u \ldots \tau_0]) \wedge D \leq \tau_0 \not\Rrightarrow D \leq u$
- $\tau - \tau_0 \leq 3$ is not pre-idle-invariant
  $u \leq \tau_0 \leq \tau \wedge \tau_0 < \infty \wedge \mathit{stable}(o, [u \ldots \tau_0]) \wedge \tau - \tau_0 \leq 3 \not\Rrightarrow$
  $\tau - u \leq 3$

If Q has no references to τ_0 then it is pre-idle-invariant.

Post-idle-invariant

A relation Q is post-idle-invariant if

$$x{:}\left[Q\right] \quad \sqsubseteq \quad x{:}\left[Q\right] \,;\, \textbf{idle}$$

Q is post-idle-invariant if for all τ_0, τ, and u,

$$\tau_0 \leq \tau \leq u < \infty \wedge \mathit{stable}(\mathit{outputs}, [\tau \mathinner{..} u]) \wedge Q \Rrightarrow Q\,[u/\tau]$$

Which of the following predicates is pre-idle-invariant?

- $\tau \leq D$
- $\tau_0 \leq D$
- $D \leq \tau$
- $\tau - \tau_0 \leq 3$

Post-idle-invariant exercises...

- $\tau \leq D$ is not post-idle-invariant
  $\tau_0 \leq \tau \leq u < \infty \wedge \mathit{stable}(o, [\tau \ldots u]) \wedge \tau \leq D \not\Rightarrow u \leq D$
- $\tau_0 \leq D$ is post-idle-invariant
  $\tau_0 \leq \tau \leq u < \infty \wedge \mathit{stable}(o, [\tau \ldots u]) \wedge \tau_0 \leq D \Rrightarrow \tau_0 \leq D$
- $D \leq \tau$ is post-idle-invariant
  $\tau_0 \leq \tau \leq u < \infty \wedge \mathit{stable}(o, [\tau \ldots u]) \wedge D \leq \tau \Rrightarrow D \leq u$
- $\tau - \tau_0 \leq 3$ is not post-idle-invariant
  $\tau_0 \leq \tau \leq u < \infty \wedge \mathit{stable}(o, [\tau \ldots u]) \wedge \tau - \tau_0 \leq 3 \not\Rightarrow$
  $u - \tau_0 \leq 3$

If Q has no references to τ then it is post-idle-invariant.

Returning to the conditional

Provided

- B is idle-stable,
- P is idle-invariant, and
- Q is both pre- and post-idle-invariant

then

$$\left\{P\right\}\, x{:}\left[Q\right] \sqsubseteq \begin{array}{l} \textbf{if}\ B\ \textbf{then}\ \left\{P \wedge B\right\}\, x{:}\left[Q\right] \\ \qquad\qquad \textbf{else}\ \left\{P \wedge \neg\, B\right\}\, x{:}\left[Q\right] \end{array}$$

The essential properties can be summarised by

$$\left\{P\right\}\, x{:}\left[Q\right] \sqsubseteq \textbf{idle};\ \left\{P\right\}\, x{:}\left[Q\right]\ ;\ \textbf{idle}$$

Example: smoke detector

A process is required to monitor a smoke detector input and when smoke is detected terminate within p seconds. If the smoke detector never triggers, the process never terminates. The smoke detector is modelled by a single boolean input, *smoke*.

input *smoke* : *boolean*;

const $p = 1\mathrm{s}$;

con *trig* : *Time*;

Assumption

If *smoke* triggers then it is assumed that it will stay triggered for at least a period of p seconds, where p is some finite constant. We use the time *trig* to stand for the exact time at which *smoke* triggers; if it never triggers then *trig* is infinity. The input *smoke* is *false* up until time *trig*, and if *trig* is not infinity *smoke* is true for at least another p seconds:

(1)
$$\left\{ \begin{array}{l} \forall\, t : Time \bullet \big(t < trig \Rightarrow \neg\, smoke(t)\big) \wedge \\ \qquad\qquad \big(trig \leq t \leq trig + p \Rightarrow smoke(t)\big) \end{array} \right\}$$

Terminating and non-terminating cases

$$(2) \qquad \left\{ \tau \leq \mathit{trig} \right\} \quad \infty \left[\begin{array}{c} (\tau < \infty \wedge \mathit{trig} \leq \tau \leq \mathit{trig} + p) \vee \\ (\tau = \infty \wedge \mathit{trig} = \infty) \end{array} \right]$$

Sequential composition

R_1 and R_2 effectively define relations between states, and we use the notation $R_1 \overset{\circ}{9} R_2$ to stand for their relational composition: a before state is related to a final state by $R_1 \overset{\circ}{9} R_2$ if there exists an intermediate state related to the before state by R_1 and which is related to the after state by R_2.

$$\left\{P_1\right\}\ x{:}\left[P_2 \wedge (R_1 \overset{\circ}{9} R_2)\right]$$

$$\sqsubseteq \left\{P_1\right\}\ x{:}\left[P_m \wedge R_1\right];\ \left\{P_m\right\}\ x{:}\left[P_2 \wedge R_2\right]$$

Possibly nonterminating

A sequential composition may not terminate either because its first command does not terminate, or because its first command terminates but its second does not.

$$\left\{P_1\right\} \infty x: \left[\begin{array}{c} (\tau < \infty \wedge P_2 \wedge (R_1 \overset{\circ}{9} R_2)) \\ \vee\ (\tau = \infty \wedge (Q_1 \vee (R_1 \overset{\circ}{9} Q_2)) \end{array}\right]$$

$$\sqsubseteq$$

$$\left\{P_1\right\} \infty x: \left[(\tau < \infty \wedge P_m \wedge R_1) \vee (\tau = \infty \wedge Q_1)\right] ;$$
$$\left\{P_m\right\} \infty x: \left[(\tau < \infty \wedge P_2 \wedge R_2) \vee (\tau = \infty \wedge Q_2)\right]$$

This law generalises that of Jones. His law is the special case when both Q_1 and Q_2 are the predicate *false*.

Separating out a deadline

$$\infty x: \left[\begin{array}{c}(\tau < \infty \wedge R \wedge P_2 \wedge \tau \leq D) \vee \\ (\tau = \infty \wedge Q)\end{array}\right]$$

$$\sqsubseteq \infty x: \left[\begin{array}{c}(\tau < \infty \wedge R \wedge P_2) \vee \\ (\tau = \infty \wedge Q)\end{array}\right] ; \infty \left[\tau_0 = \tau \wedge \tau \leq D\right]$$

$$\sqsubseteq \infty x: \left[\begin{array}{c}(\tau < \infty \wedge R \wedge P_2) \vee \\ (\tau = \infty \wedge Q)\end{array}\right] ; \textbf{deadline}\ D$$

Example: separating the deadline

The postcondition of the specification (2) contains a deadline, $\tau \leq \mathit{trig} + p$, that we can separate out.

$$\left\{\tau \leq \mathit{trig}\right\} \infty \left[\begin{array}{c}(\tau < \infty \wedge \mathit{trig} \leq \tau \leq \mathit{trig} + p) \vee \\ (\tau = \infty \wedge \mathit{trig} = \infty)\end{array}\right]$$

$$\sqsubseteq \left\{\tau \leq \mathit{trig}\right\} \infty \left[\begin{array}{c}(\tau < \infty \wedge \mathit{trig} \leq \tau) \vee \\ (\tau = \infty \wedge \mathit{trig} = \infty)\end{array}\right]; \qquad (3)$$

deadline $\mathit{trig} + p$

Note that if the first command (3) never terminates, the deadline command is never reached, i.e., the deadline only needs to be met when the first command terminates.

Repetition

The standard law makes use of an invariant I and a well-founded relation R.

$$\{I\};\ x{:}\left[\neg B \wedge I \wedge R^*\right] \quad \sqsubseteq \quad \textbf{do}\, B \to \{B \wedge I\};\ x{:}\left[I \wedge R\right]\ \textbf{od}$$

Nonterminating repetition law

The law for introducing a repetition assumes the following are given: an idle-stable, boolean-valued expression, B—the guard; a single-state, idle-invariant predicate, I—the loop invariant; a pre-idle-invariant relation R'; and a weaker pre- and post-idle-invariant relation R. Here we restrict ourselves to a repetition whose body is guaranteed to terminate on every iteration.

$$\left\{ I \right\};\ \infty x\colon \left[\begin{array}{c} (\tau < \infty \land \neg\, B \mathbin{@} \tau \land I \land R^*) \lor \\ (\tau = \infty \land (\exists\, d : \mathit{Time} \bullet 0 < d \land \\ (\tau_0 + d \leq \tau \land B \mathbin{@} \tau_0 \land I \mathbin{@} \tau_0 \land R')^\infty)) \end{array} \right]$$

$$\sqsubseteq$$

$$\mathbf{do}\, B \to \left\{ B \mathbin{@} \tau \land I \right\};\ x\colon \left[I \land R' \right]\ \mathbf{od}$$

Infinite iteration

This is a relation composed with itself infinitely many times. To define this we make use of infinite sequences $\vec{t}$ and $\vec{v}$ corresponding to the sequences of intermediate times and variable values between each composition of the relation.

$$R^{\infty} \;\widehat{=}\; (\exists \vec{t} : \mathbb{N} \to \mathit{Time};\ \vec{v} : \mathbb{N} \to T_v \bullet \vec{t}(0) = \tau_0 \land \vec{v}(0) = v_0 \land (\forall i : \mathbb{N} \bullet R \left[\frac{\vec{t}(i), \vec{t}(i+1), \vec{v}(i), \vec{v}(i+1)}{\tau_0, \tau, v_0, v} \right]))$$

If R is well founded then R^{∞} is false, and the repetition law reduces to the terminating one.

Example: a nonterminating repetition

In order to develop a repetition that refines the specification (3), we need to find a guard B, an invariant I, and relation R' and a weaker relation R, such that both of the following hold.

$$\text{(4)} \qquad \neg\, B \,@\, \tau \wedge I \wedge R^* \;\Rightarrow\; trig \leq \tau$$

$$\text{(5)} \qquad \left(\begin{array}{l} \exists\, d : Time \bullet 0 < d \wedge \\ \quad (\tau_0 + d \leq \tau \wedge B \,@\, \tau_0 \wedge I \,@\, \tau_0 \wedge R')^\infty \end{array} \right) \;\Rightarrow\; trig = \infty$$

We introduce a boolean variable tr, which is used to sample the value of *smoke*. If tr is true then the time is after *trig*; hence the invariant has as a conjunct

$$tr \Rightarrow trig \leq \tau$$

Auxiliary variable sample

If the repetition has a guard of $\neg\ tr$, the negation of the guard and this invariant are enough to establish (4). For this example R is just $true$.
If tr is false then $trig$ is later than the time of the most recent sample. We introduce an auxiliary variable $sample$ to stand for (a lower bound on) this time. The invariant includes the conjunct

$$\neg\ tr \Rightarrow sample \leq trig$$

Both conjuncts of the invariant are established by initial assignments before the repetition consisting of

$$sample := \tau;$$

$$tr := false$$

Upper bound on sample time

Because the smoke detector is only guaranteed to stay *true* for p seconds, it must be sampled within p seconds of the previous sample in order not to be missed. The relation R' puts an upper bound on the time of completion of the body of the repetition in terms of the time of the previous sample, $sample_0$, i.e., the value of *sample* at the start of an iteration.

$$R' \;\widehat{=}\; \tau \leq sample_0 + p$$

The nontermination case

If we combine $B \mathbin{@} \tau_0 \land I \mathbin{@} \tau_0 \land R'$ we get

$$\neg\, tr_0 \land sample_0 \leq trig \land \tau \leq sample_0 + p$$

This implies $\tau \leq trig + p$, and hence the infinite iteration in the postcondition of the law above implies

$$\vec{t}(0) = \tau_0 \land (\forall\, i : \mathbb{N} \bullet \vec{t}(i) + d \leq \vec{t}(i+1) \land \vec{t}(i+1) \leq trig + p)$$

The minimum separation between successive elements of $\vec{t}$ guarantees that its values are unbounded, and because $trig + p$ is greater than all elements of $\vec{t}$ (and p is a finite constant), that $trig$ must be infinity.

The refined program

The code developed so far is

$$\left\{\tau \leq \mathit{trig}\right\};$$

$$\mathit{sample} := \tau;$$

$$\mathit{tr} := \mathit{false};$$

$$\textbf{do}\ \neg\ \mathit{tr}\ \rightarrow$$

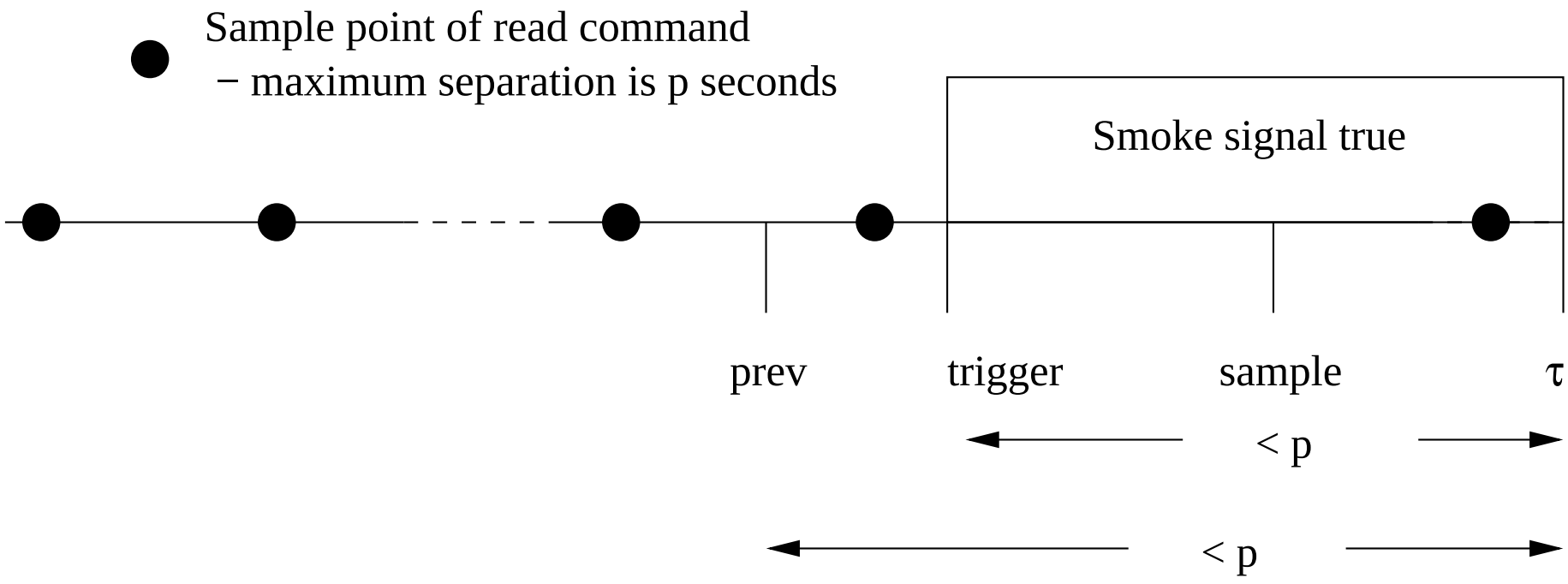
$$\mathit{tr}, \mathit{sample}: \left[\neg\ \mathit{tr}\ \wedge\ I, \quad I\ \wedge\ \tau \leq \mathit{sample}_0 + p\right] \tag{6}$$

$$\textbf{od};$$

$$\textbf{deadline}\ \mathit{trig} + p$$

where $I \mathrel{\widehat{=}} (\mathit{tr} \Rightarrow \mathit{trig} \leq \tau) \wedge (\neg\ \mathit{tr} \Rightarrow \mathit{sample} \leq \mathit{trig})$.

Sampling of smoke detector



Timing constraint analysis

$$\mathsf{A} :: \left\{ \tau \leq \mathit{trig} \right\};$$

$\mathit{sample} := \tau;$

$\mathit{tr} := \mathit{false};$

do $\neg\, \mathit{tr} \rightarrow$

$\mathsf{B} :: \mathit{prev}, \mathit{sample} := \mathit{sample}, \tau;$

$\mathit{tr} : \mathbf{read}(\mathit{smoke});$

$\mathsf{C} :: \mathbf{deadline}\ \mathit{prev} + p;$

$$\left\{ (\mathit{tr} \Rightarrow \mathit{trig} \leq \tau) \wedge (\neg\, \mathit{tr} \Rightarrow \mathit{sample} \leq \mathit{trig}) \right\}$$

od;

$\mathsf{D} :: \mathbf{deadline}\ \mathit{trig} + p$

Path entering the loop

The first path we consider is the path that enters the loop and reaches the deadline in the loop for the first time.

$$\mathsf{A} :: \left\{\tau \leq \mathit{trig}\right\};$$
$$\mathit{sample} := \tau;$$
$$\mathit{tr} := \mathit{false};$$
$$\left[\neg\, \mathit{tr}\right]$$
$$\mathsf{B} :: \mathit{prev}, \mathit{sample} := \mathit{sample}, \tau;$$
$$\mathit{tr} : \mathbf{read}\left(\mathit{smoke}\right);$$
$$\mathsf{C} :: \mathbf{deadline}\, \mathit{prev} + p$$

Non-initial iterations

To ensure the deadline in the loop is reached on time on non-initial iterations, we consider the path that starts at (B), executes the body of the repetition, restarts the repetition because the guard holds, and executes down to (C).

B :: $prev, sample := sample, \tau$;

tr : **read**$(smoke)$;

C :: **deadline** $prev + p$;

$\left\{ (tr \Rightarrow trig \leq \tau) \wedge (\neg\, tr \Rightarrow sample \leq trig) \right\}$

$[\neg\, tr]$

B :: $prev, sample := sample, \tau$;

tr : **read**$(smoke)$;

C :: **deadline** $prev + p$

Exit path from repetition

B :: $prev, sample := sample, \tau$;

tr : **read**$(smoke)$;

C :: **deadline** $prev + p$;

$\{(tr \Rightarrow trig \leq \tau) \wedge (\neg\, tr \Rightarrow sample \leq trig)\}$

$[\neg\, tr]$

B :: $prev, sample := sample, \tau$;

tr : **read**$(smoke)$;

C :: **deadline** $prev + p$;

$\{(tr \Rightarrow trig \leq \tau) \wedge (\neg\, tr \Rightarrow sample \leq trig)\}$

$[tr]$;

D :: **deadline** $trig + p$

The General Refinement Law

We assume the body of the repetition consists of a deadline command followed by a specification.

$$\begin{array}{ll} \textbf{do}\ B \rightarrow & \textbf{deadline}\ D; \\ & \left\{ B \mathbin{@} \tau \wedge \tau \leq D \mathbin{@} \tau \wedge I \right\}; \\ & \infty x : \left[(\tau < \infty \wedge I \wedge R) \vee (\tau = \infty \wedge Q) \right] \\ \textbf{od} & \end{array}$$

The specification can assume that the initial time is before the deadline and that both the guard and the invariant hold initially. The body of the repetition either terminates, reestablishing I and establishing R between its initial and final states, or it fails to terminate but establishes Q.

Repetition

This repetition refines a specification that assumes that the invariant holds initially and either,

- terminates in a state in which the guard is false, the invariant holds, and the relation R^* is established between the initial and final states;
- fails to terminate because the body failed to terminate, and overall establishes $(R^* \stackrel{\mathrm{Q}}{9} Q)$; or
- fails to terminate because the body always terminates but the guard always remains true, in which case the predicate I_∞ and infinite iteration of the relation, R^∞, are established.

Repetition law

Law 3 (repetition) *Given an idle-stable, boolean-valued expression, B; a single-state, idle-invariant predicate, I; an idle-stable, time-valued expression, D; a pre-idle-invariant relation Q; and a pre- and post-idle invariant relation R; then*

$$\left\{I\right\};\ \infty x\colon \left[\begin{array}{c}(\tau < \infty \wedge \neg\, B \mathbin{@} \tau \wedge I \wedge R^*) \vee \\ (\tau = \infty \wedge ((I_\infty \wedge R^\infty) \vee (R^* \mathbin{{}^{0}_{9}} Q)))\end{array}\right]$$

$$\sqsubseteq \textbf{do}\ B \rightarrow\ \textbf{deadline}\ D;\ \left\{B \mathbin{@} \tau \wedge \tau \leq D \mathbin{@} \tau \wedge I\right\};$$

$$\infty x\colon \left[(\tau < \infty \wedge I \wedge R) \vee (\tau = \infty \wedge Q)\right]$$

od

where $I_\infty \mathrel{\widehat{=}} (\forall\, t : \mathit{Time} \bullet (\exists\, \tau : \mathit{Time};\ v : T_v \bullet t \leq \tau \wedge B \mathbin{@} \tau \wedge \tau \leq D \mathbin{@} \tau \wedge I \wedge R^*))$

Special cases of the law

Taking Q as the predicate *false* gives the following special case.

Law 4 (repetition–terminating body) *Given an idle-stable boolean-valued expression, B; a single-state idle-invariant predicate, I; a pre- and post-idle-invariant relation R; and an idle-stable, time-valued expression, D; then*

$$\left\{ I \right\};\ \infty x\colon \left[\begin{array}{c} (\tau < \infty \wedge \neg\, B \mathbin{@} \tau \wedge I \wedge R^{*}) \vee \\ (\tau = \infty \wedge I_{\infty} \wedge R^{\infty}) \end{array} \right]$$

$$\sqsubseteq\ \textbf{do}\ B \rightarrow\ \ \textbf{deadline}\ D;\ \left\{ B \mathbin{@} \tau \wedge \tau \leq D \mathbin{@} \tau \wedge I \right\};\ x\colon \left[I \wedge R \right]$$

od

Terminating repetition

If R is a well-founded relation then $R^\infty \equiv \mathit{false}$, and hence the infinite iteration alternative is ruled out. In addition, if D is infinity, the deadline introduces no constraint whatsoever (**deadline** $\infty \sqsubseteq$ **skip**) and the law reduces to the following.

Law 5 (terminating repetition) *Given an idle-stable, boolean-valued expression, B; a single-state, idle-invariant predicate, I; and a pre- and post-idle-invariant, well-founded relation R; then*

$$\begin{array}{l} \quad \left\{ I \right\};\ x{:}\left[\neg\, B\ @\,\tau \wedge I \wedge R^* \right] \\ \sqsubseteq\ \mathbf{do}\, B \rightarrow \left\{ B\ @\,\tau \wedge I \right\};\ x{:}\left[I \wedge R \right]\ \mathbf{od} \end{array}$$

This is the standard law for refinement to a terminating repetition given in the relational form.

True guard

A repetition with a constant true guard never terminates.

Law 6 (repetition–true guard) *Given a single-state, idle-invariant predicate, I; an idle-stable, time-valued expression, D; a pre-idle-invariant relation Q; and a pre- and post-idle invariant relation R; then*

$$\left\{ I \right\};\ \infty x\colon \left[(\tau = \infty \wedge ((I_\infty \wedge R^\infty) \vee (R^* \,{}^{\mathrm{o}}_{9}\, Q))) \right]$$

$$\sqsubseteq \ \textbf{do}\ \mathit{true} \rightarrow \ \textbf{deadline}\ D;$$

$$\left\{ \tau \leq D \mathbin{@} \tau \wedge I \right\};$$

$$\infty x\colon \left[(\tau < \infty \wedge I \wedge R) \vee (\tau = \infty \wedge Q) \right]$$

od

True guard and terminating body

A special case of this law is for an always terminating body. As with Law 4 this can be handled by choosing Q to be *false*.

Law 7 (repetition–true guard and terminating body) *Given a single-state, idle-invariant predicate, I; an idle-stable, time-valued expression, D; and a pre- and post-idle invariant relation R; then*

$$\left\{ I \right\};\ \infty x{:}\left[\tau = \infty \wedge \left(I_\infty \wedge R^\infty \right) \right]$$

$$\sqsubseteq\ \textbf{do}\ \mathit{true} \rightarrow\ \ \textbf{deadline}\ D;$$

$$\left\{ \tau \leq D \mathbin{@} \tau \wedge I \right\};\ \infty x{:}\left[\tau < \infty \wedge I \wedge R \right]$$

od

Deadline as termination

For a repetition with a terminating body ($Q \equiv \mathit{false}$), if the deadline D is constant for the duration of the entire repetition, then the repetition is guaranteed to terminate. This follows because I_∞ is false, due to the fact that τ' cannot be less than the fixed value D for all times τ'.

Law 8 (deadline as termination) *Given an idle-stable boolean-valued expression, B; a single-state, idle-invariant predicate, I; a pre- and post-idle-invariant relation R; and an idle-stable, time-valued expression, D, which does not include any references to variables in the frame; then*

$$\left\{I \wedge D \mathbin{@} \tau < \infty\right\};\ x{:}\left[\neg\, B \mathbin{@} \tau \wedge I \wedge R^*\right]$$
$$\sqsubseteq \textbf{do}\, B \to \textbf{deadline}\, D;\ \left\{B \mathbin{@} \tau \wedge \tau \leq D \mathbin{@} \tau \wedge I\right\};\ x{:}\left[I \wedge R\right]\ \textbf{od}$$

Conclusions

- Machine independence
  - The deadline command
  - Idle-stable expressions
  - Idle-invariant predicates
  - Auxiliary variables
- Nonterminating processes
  - Sequential composition for nonterminating commands
  - Nonterminating repetitions
- Timing path analysis

References

[Heh89] E. C. R. Hehner. Termination is timing. In J.L.A. van de Snepscheut, editor, *Mathematics of Program Construction*, volume 375 of *Lecture Notes in Computer Science*, pages 36–47. Springer, June 1989.

[Heh93] E. C. R. Hehner. *A Practical Theory of Programming*. Springer, 1993.

[HH98] C. A. R. Hoare and Jifeng He. *Unifying Theories of Programming*. Prentice Hall, 1998.