Compositional Action System Derivation Using Enforced Properties

Brijesh Dongol and Ian J. Hayes

School of Information Technology and Electrical Engineering,
The University of Queensland

MPC 2010

Outline

History

- Sequential program derivation (Dijkstra, 1975)
- Safety-based concurrent program derivation (Feijen and van Gasteren, 1999)
- Progress-based concurrent program derivation (Dongol and Mooij, 2006 & 2008)
- Enforced properties (Dongol and Hayes, 2009), (Dongol, 2009) - PhD thesis

This paper

	Goals	Techniques
1.	Calculational development	
2.	Incremental trace refinement	
3.	Compositionally address safety and progress	
4.	Simplify existing rules	

This paper

	Goals	Techniques
1.	Calculational development	Weakest precondition
2.	Incremental trace refinement	
3.	Compositionally address safety and progress	
4.	Simplify existing rules	

This paper

	Goals	Techniques
1.	Calculational development	Weakest precondition
2.	Incremental trace refinement	Enforced properties and frames
3.	Compositionally address safety and progress	
4.	Simplify existing rules	

This paper

	Goals	Techniques
1.	Calculational development	Weakest precondition
2.	Incremental trace refinement	Enforced properties and frames
3.	Compositionally address safety and progress	Temporal logic on relations
4.	Simplify existing rules	

This paper

	Goals	Techniques
1.	Calculational development	Weakest precondition
2.	Incremental trace refinement	Enforced properties and frames
3.	Compositionally address safety and progress	Temporal logic on relations
4.	Simplify existing rules	Action systems framework

Syntax

Statements and actions

$$
\begin{array}{lcl}
S & ::= & \mathbf{diverge} \mid \mathbf{skip} \mid x := E \mid x :\in V \mid S_1 \,;\, S_2 \mid x \cdot [\![S]\!] \\
A & ::= & b \longrightarrow S \mid A_1 \sqcap A_2 \mid x \cdot [\![A]\!]
\end{array}
$$

Action systems

$\mathscr{A} \mathrel{\hat{=}} A_0 \,;\, \mathbf{do}\ A\ \mathbf{od}$ is an action system with initialisation action A_0 and main action A

Frames

- If x is a variable of type T:

$$\begin{aligned} x \cdot [\![S]\!] &= S \,;\, x :\in T \\ x \cdot [\![b \longrightarrow S]\!] &= b \longrightarrow x \cdot [\![S]\!] \\ x \cdot [\![A_1 \sqcap A_2]\!] &= x \cdot [\![A_1]\!] \sqcap x \cdot [\![A_2]\!] \end{aligned}$$

Frames

- If x is a variable of type T:

$$\begin{aligned} x \cdot [\![S]\!] &= S \,;\, x :\in T \\ x \cdot [\![b \longrightarrow S]\!] &= b \longrightarrow x \cdot [\![S]\!] \\ x \cdot [\![A_1 \sqcap A_2]\!] &= x \cdot [\![A_1]\!] \sqcap x \cdot [\![A_2]\!] \end{aligned}$$

- Frames allow introduction of new (internal) variables

Frames

- If x is a variable of type T:

$$\begin{aligned} x \cdot [\![S]\!] &= S \,;\, x :\in T \\ x \cdot [\![b \longrightarrow S]\!] &= b \longrightarrow x \cdot [\![S]\!] \\ x \cdot [\![A_1 \sqcap A_2]\!] &= x \cdot [\![A_1]\!] \sqcap x \cdot [\![A_2]\!] \end{aligned}$$

- Frames allow introduction of new (internal) variables
- Frames can be turned into statements by refinement

Refinement

Suppose $s \in \text{seq}.\Sigma$. We assume

- $rL.s$ *removes local variables* from each state in s
- $rS.s$ *removes stuttering* in s

Definition (Trace refinement)

$$\mathscr{A} \sqsubseteq_{Tr} \mathscr{C} \quad \hat{=} \quad \forall t : Tr.\mathscr{C} \bullet \exists s : Tr.\mathscr{A} \bullet rS.(rL.s) = rS.(rL.t)$$

Refinement

Suppose $s \in \mathrm{seq}.\Sigma$. We assume

- $rL.s$ *removes local variables* from each state in s
- $rS.s$ *removes stuttering* in s

Definition (Trace refinement)

$$\mathscr{A} \sqsubseteq_{Tr} \mathscr{C} \quad \hat{=} \quad \forall t : Tr.\mathscr{C} \bullet \exists s : Tr.\mathscr{A} \bullet rS.(rL.s) = rS.(rL.t)$$

Lemma

If $Tr.\mathscr{C} \subseteq Tr.\mathscr{A}$, *then* $\mathscr{A} \sqsubseteq_{Tr} \mathscr{C}$.

A temporal logic on relations

Linear temporal logic (LTL)

(Manna and Pnueli, 1992)
always ($\Box$), eventually ($\Diamond$), until ($\mathcal{U}$), unless ($\mathcal{W}$) defined for formulas on single-state predicates

Relational linear temporal logic (RLTL)

Defined over two-state relations

- Semantics mostly the same as LTL
- Difference: For sequence of states s and RLTL formula Q,

$$(s,u) \vdash \Box Q \iff \forall v : \mathrm{dom}.s \bullet v \geq u \wedge v+1 \in \mathrm{dom}.s \Rightarrow (s,v) \vdash Q$$

Enforced properties

Definition (Enforced property)

Suppose, $\mathscr{A}$ is an action system and R is a RLTL formula. Action system $\mathscr{A}$ with *enforced property* R, denoted $\mathscr{A}?R$, is an action system such that

$$Tr.(\mathscr{A}?R) \mathrel{\hat{=}} \{s : Tr.\mathscr{A} \mid s \vdash R\}$$

Enforced properties

- Given an initial action system $\mathscr{A}$, suppose we want to derive an action system that satisfies RLTL property R

Enforced properties

- Given an initial action system $\mathscr{A}$, suppose we want to derive an action system that satisfies RLTL property R
  1. Enforce property R on $\mathscr{A}$ to obtain $\mathscr{A}?R$

Enforced properties

- Given an initial action system $\mathscr{A}$, suppose we want to derive an action system that satisfies RLTL property R
  1. Enforce property R on $\mathscr{A}$ to obtain $\mathscr{A}?R$
  2. Derive $\mathscr{B}$ such that $\mathscr{A}?R \sqsubseteq_{Tr} \mathscr{B}$ and $Tr.\mathscr{B} \models R$

Enforced properties

- Given an initial action system $\mathscr{A}$, suppose we want to derive an action system that satisfies RLTL property R
  1. Enforce property R on $\mathscr{A}$ to obtain $\mathscr{A}?R$
  2. Derive $\mathscr{B}$ such that $\mathscr{A}?R \sqsubseteq_{Tr} \mathscr{B}$ and $Tr.\mathscr{B} \models R$
  3. Hence, $\mathscr{B}$ satisfies R and furthermore does not need to enforce R

Enforced properties

- Given an initial action system $\mathscr{A}$, suppose we want to derive an action system that satisfies RLTL property R
  1. Enforce property R on $\mathscr{A}$ to obtain $\mathscr{A}?R$
  2. Derive $\mathscr{B}$ such that $\mathscr{A}?R \sqsubseteq_{Tr} \mathscr{B}$ and $Tr.\mathscr{B} \models R$
  3. Hence, $\mathscr{B}$ satisfies R and furthermore does not need to enforce R
- We have rules that allow
  - introduction of new enforced properties
  - manipulation of existing enforced properties
  - introduction of new variables (via frames)
  - introduction and modification of actions

Enforced properties

- Given an initial action system $\mathscr{A}$, suppose we want to derive an action system that satisfies RLTL property R
  1. Enforce property R on $\mathscr{A}$ to obtain $\mathscr{A}?R$
  2. Derive $\mathscr{B}$ such that $\mathscr{A}?R \sqsubseteq_{Tr} \mathscr{B}$ and $Tr.\mathscr{B} \models R$
  3. Hence, $\mathscr{B}$ satisfies R and furthermore does not need to enforce R
- We have rules that allow
  - introduction of new enforced properties
  - manipulation of existing enforced properties
  - introduction of new variables (via frames)
  - introduction and modification of actions
- Each rule ensures trace refinement

Enforced properties

Lemma

For action systems $\mathscr{A}$ and $\mathscr{C}$, and RLTL formulae R and R' each of the following holds:

a.	$\mathscr{A}$	$\sqsubseteq_{Tr}$	$\mathscr{A}?R$	
b.	$\mathscr{A}?R$	$\sqsubseteq_{Tr}$	$\mathscr{A}?R'$	*provided* $R' \Rightarrow R$
c.	$\mathscr{A}?R$	$\sqsubseteq_{Tr}$	$\mathscr{C}?R$	*provided* $\mathscr{A} \sqsubseteq_{Tr} \mathscr{C}$
d.	$\mathscr{A}?R$	$\sqsupseteq_{Tr}$	$\mathscr{A}$	*provided* $Tr.\mathscr{A} \models R$
e.	$\mathscr{A}?(R \wedge R')$	$\sqsupseteq_{Tr}$	$(\mathscr{A}?R)?R'$	

Enforced properties

Lemma

For action systems $\mathscr{A}$ and $\mathscr{C}$, and RLTL formulae R and R' each of the following holds:

a.	$\mathscr{A}$	$\sqsubseteq_{Tr}$	$\mathscr{A}?R$	
b.	$\mathscr{A}?R$	$\sqsubseteq_{Tr}$	$\mathscr{A}?R'$	*provided* $R' \Rightarrow R$
c.	$\mathscr{A}?R$	$\sqsubseteq_{Tr}$	$\mathscr{C}?R$	*provided* $\mathscr{A} \sqsubseteq_{Tr} \mathscr{C}$
d.	$\mathscr{A}?R$	$\sqsupseteq_{Tr}$	$\mathscr{A}$	*provided* $Tr.\mathscr{A} \models R$
e.	$\mathscr{A}?(R \wedge R')$	$\sqsupseteq_{Tr}$	$(\mathscr{A}?R)?R'$	

Proof.

If $Tr.\mathscr{C} \subseteq Tr.\mathscr{A}$, then $\mathscr{A} \sqsubseteq_{Tr} \mathscr{C}$ (from previous lemma). □

Enforced properties

Definition

If A is an action, p is a predicate, and r a relation, then

$$\begin{array}{lcl} A!p & \widehat{=} & [p]\,;A\,;[p] \\ A!r & \widehat{=} & \langle rel.A \cap r \rangle \end{array}$$

- $A!p$ blocks if
  - p does not hold prior to executing A, or
  - the execution of A does not establish p
- $A!r$ blocks if no execution of A can satisfy r

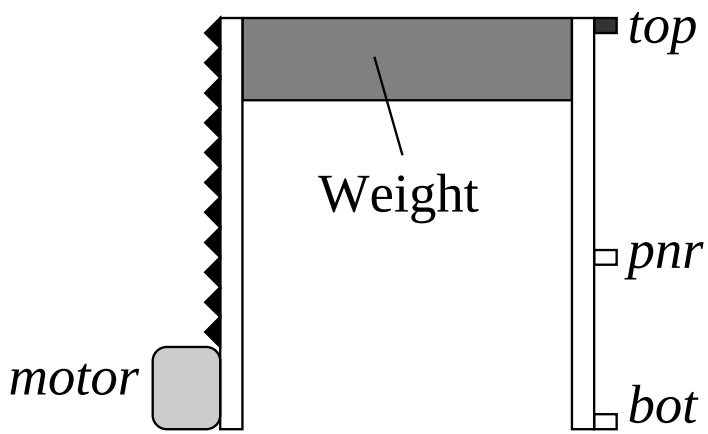
Enforced properties

Lemma

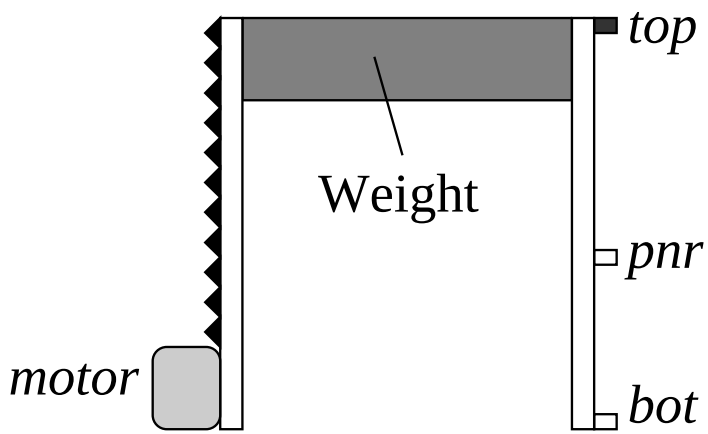
For actions A, A_1, and A_2; predicates p, p_1, and p_2; and relations r, r_1, and r_2, each of the following holds:

a.	$A!(p_1 \wedge p_2)$	$=$	$(A!p_1)!p_2$	
b.	$A!(r_1 \cap r_2)$	$=$	$(A!r_1)!r_2$	
c.	$A!p$	$=$	$[p]\,;A$	*provided* $p \Rightarrow A.p$
d.	$A!r$	$=$	A	*provided* $rel.A \subseteq r$
e.	$(A_1 \sqcap A_2)!p$	$=$	$(A_1!p) \sqcap (A_2!p)$	
f.	$(A_1 \sqcap A_2)!r$	$=$	$(A_1!r) \sqcap (A_2!r)$	

Example - Industrial press



Example - Industrial press



- Program variables: *locked* and *motor*
- Environment variables: *top*, *pnr*,*bot*, and *pressed*

Requirements

Safety requirements:

$$\Box(locked \Rightarrow top \wedge motor = Off) \tag{1}$$

$$\Box(locked \Rightarrow (locked \,\mathcal{W}\, pressed)) \tag{2}$$

$$\Box(\neg locked \wedge motor = Off \Rightarrow (\neg locked \wedge motor = Off)\ \mathcal{W}\ ((bot \vee \neg pnr) \wedge \neg pressed)) \tag{3}$$

$$\Box(pnr \wedge motor = Off \Rightarrow ((pnr \wedge motor = Off)\ \mathcal{W}\ (bot \wedge \neg pressed))) \tag{4}$$

$$\Box(motor = On \Rightarrow (motor = On\ \mathcal{W}\ locked)) \tag{5}$$

Progress requirements: we say $p \leadsto q \iff \Box(p \Rightarrow \Diamond q)$

$$(locked \wedge pressed) \leadsto \neg locked \tag{6}$$

$$(\neg locked \wedge \neg pnr \wedge \neg pressed) \leadsto motor = On \tag{7}$$

$$(bot \wedge \neg pressed) \leadsto motor = On \tag{8}$$

Assumptions on the environment

Safety assumptions:

$$(top \Rightarrow \neg pnr \wedge \neg bot) \wedge (bot \Rightarrow pnr) \tag{9}$$

$$top, pnr, bot \mid \begin{array}{l} (top \Rightarrow \neg pnr' \wedge \neg bot') \wedge (\neg top \wedge \neg pnr \Rightarrow \neg bot') \\ (pnr \Rightarrow \neg top') \wedge (bot \Rightarrow pnr') \end{array} \tag{10}$$

Progress assumptions:

$$\Diamond\Box(\neg locked \wedge motor = Off) \Rightarrow \Box\Diamond\neg top \wedge \Box\Diamond pnr \wedge \Box\Diamond bot \tag{11}$$

$$\Diamond\Box(motor = On) \Rightarrow \Box\Diamond top \wedge \Box\Diamond\neg bot \wedge \Box\Diamond\neg pnr \tag{12}$$

Initial program

Define:

$$\begin{aligned} \mathit{Safe} &\mathrel{\widehat{=}} (1) \wedge (2) \wedge (3) \wedge (4) \wedge (5) \\ \mathit{Prog} &\mathrel{\widehat{=}} (6) \wedge (7) \wedge (8) \\ \mathit{RelyProg} &\mathrel{\widehat{=}} (11) \wedge (12) \end{aligned}$$

$$\mathit{env} \mathrel{\widehat{=}} \mathit{pressed} \cdot [\![\langle (10) \rangle]\!] ! (9)$$

$$\begin{array}{l} \mathbf{do}\ \ \mathit{true} \longrightarrow \mathit{env} \\ \sqcap\ \ \mathit{true} \longrightarrow \mathit{motor}, \mathit{locked} \cdot [\![\mathbf{skip}]\!] \\ \mathbf{od} \\ ?(\mathit{Safe} \wedge \mathit{Prog} \wedge \mathit{RelyProg}) \end{array}$$

Calculational rules

Lemma (Unless)

$\Box(p \Rightarrow (p \mathcal{W} q))$ *holds provided* $\Box(p \wedge \neg q \Rightarrow p' \vee q')$

Derivation - replace unless properties

$$
\begin{array}{ll}
 & (2) \\
= & \quad \{\text{definition}\} \\
 & \Box(\mathit{locked} \Rightarrow \mathit{locked}\ \mathcal{W}\ \mathit{pressed}) \\
\Leftarrow & \quad \{\text{Lemma (unless)}\} \\
 & \Box(\mathit{locked} \wedge \neg \mathit{pressed} \Rightarrow (\mathit{locked}' \vee \mathit{pressed}')) \\
\Leftarrow & \quad \{\text{logic}\} \\
 & \Box(\mathit{locked} \wedge \neg \mathit{locked}' \Rightarrow \mathit{pressed})
\end{array}
$$

- Apply similar calculation to (3), (4), (5) to obtain $Safe'$.
- $Safe' \Rightarrow Safe$
- Recall $\mathscr{A}?R \sqsubseteq_{Tr} \mathscr{A}?R'$ provided $R' \Rightarrow R$

- Apply similar calculation to (3), (4), (5) to obtain $Safe'$.
- $Safe' \Rightarrow Safe$
- Recall $\mathscr{A}?R \sqsubseteq_{Tr} \mathscr{A}?R'$ provided $R' \Rightarrow R$

$$
\begin{array}{ll}
\textbf{do} & true \longrightarrow env \\
\sqcap & true \longrightarrow motor, locked \cdot [\![\textbf{skip}]\!] \\
\textbf{od} & \\
?(Safe \wedge Prog \wedge RelyProg) &
\end{array}
$$

- Apply similar calculation to (3), (4), (5) to obtain $Safe'$.
- $Safe' \Rightarrow Safe$
- Recall $\mathscr{A}?R \sqsubseteq_{Tr} \mathscr{A}?R'$ provided $R' \Rightarrow R$

$$
\begin{array}{l}
\textbf{do}\ \ true \longrightarrow env \\
\sqcap\ \ true \longrightarrow motor, locked \cdot [\![\textbf{skip}]\!] \\
\textbf{od} \\
?(Safe \wedge Prog \wedge RelyProg)
\end{array}
$$

$$\sqsubseteq_{Tr}$$

$$
\begin{array}{l}
\textbf{do}\ \ true \longrightarrow env \\
\sqcap\ \ true \longrightarrow motor, locked \cdot [\![\textbf{skip}]\!] \\
\textbf{od} \\
?(Safe' \wedge Prog \wedge RelyProg)
\end{array}
$$

Deriving actions

- Consider action that turns the motor on

Deriving actions

- Consider action that turns the motor on
- From calculation $wp.(motor := On).Safe'$, we obtain:

$$(motor = Off \Rightarrow (bot' \vee \neg pnr') \wedge \neg pressed') \wedge \neg locked$$

Deriving actions

- Consider action that turns the motor on
- From calculation $wp.(motor := On).Safe'$, we obtain:

$$(motor = Off \Rightarrow (bot' \vee \neg pnr') \wedge \neg pressed') \wedge \neg locked$$

- Because controller does not modify bot, pnr, and $pressed$ we get action:

$$motor = Off \wedge (bot \vee \neg pnr) \wedge \neg pressed \wedge \neg locked \rightarrow motor := On$$

Deriving actions

- Consider action that turns the motor on
- From calculation $wp.(motor := On).Safe'$, we obtain:

$$(motor = Off \Rightarrow (bot' \vee \neg pnr') \wedge \neg pressed') \wedge \neg locked$$

- Because controller does not modify bot, pnr, and $pressed$ we get action:

$$motor = Off \wedge (bot \vee \neg pnr) \wedge \neg pressed \wedge \neg locked \rightarrow motor := On$$

- Similarly, we get for actions that turn motor off and modify lock

$$motor = On \wedge top \rightarrow motor, locked := Off, true$$
$$locked \wedge motor = Off \wedge pressed \rightarrow locked := false$$

Deriving actions

- Consider action that turns the motor on
- From calculation $wp.(motor := On).Safe'$, we obtain:

$$(motor = Off \Rightarrow (bot' \vee \neg pnr') \wedge \neg pressed') \wedge \neg locked$$

- Because controller does not modify bot, pnr, and $pressed$ we get action:

$$motor = Off \wedge (bot \vee \neg pnr) \wedge \neg pressed \wedge \neg locked \rightarrow motor := On$$

- Similarly, we get for actions that turn motor off and modify lock

$$motor = On \wedge top \rightarrow motor, locked := Off, true$$
$$locked \wedge motor = Off \wedge pressed \rightarrow locked := false$$

- These actions are guaranteed to satisfy $Safe'$

Action introduction

Lemma

If $grd.(A!p) \Rightarrow grd.(A!p) \wedge b$

$$\begin{aligned} & \mathbf{do}\ A \sqcap (true \longrightarrow x \cdot [\![\mathbf{skip}]\!])\ \mathbf{od}\ ?\Box p \\ \sqsubseteq\ & \mathbf{do}\ A \sqcap (b \longrightarrow x := v)\ \mathbf{od}\ ?\Box p \end{aligned}$$

Action introduction

Recalling that $\mathscr{A}?R \sqsubseteq_{Tr} \mathscr{C}?R$ provided $\mathscr{A} \sqsubseteq_{Tr} \mathscr{C}$, we obtain:

$$
\begin{array}{l}
\textbf{do}\ \ true \longrightarrow env \\
\sqcap\ \ true \longrightarrow motor, locked \cdot [\![\textbf{skip}]\!] \\
\textbf{od} \\
?(Safe' \wedge Prog \wedge RelyProg)
\end{array}
$$

Action introduction

Recalling that $\mathscr{A}?R \sqsubseteq_{Tr} \mathscr{C}?R$ provided $\mathscr{A} \sqsubseteq_{Tr} \mathscr{C}$, we obtain:

$$
\begin{array}{l}
\textbf{do}\ \ true \longrightarrow env \\
\sqcap\ \ true \longrightarrow motor, locked \cdot [\![\textbf{skip}]\!] \\
\textbf{od} \\
?(Safe' \wedge Prog \wedge RelyProg)
\end{array}
$$

$\sqsubseteq_{Tr}$

$$
\begin{array}{l}
\textbf{do}\ \ true \longrightarrow env \\
\sqcap\ \ motor = Off \wedge (bot \vee \neg pnr) \wedge \neg pressed \wedge \neg locked \longrightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad motor := On \\
\sqcap\ \ motor = On \wedge top \longrightarrow motor, locked := Off, true \\
\sqcap\ \ locked \wedge motor = Off \wedge pressed \longrightarrow locked := false \\
\textbf{od} \\
?(Prog \wedge RelyProg) \\
\checkmark Safe'
\end{array}
$$

Conclusion

- In this paper:
  - Rules for calculational proofs of temporal formulae
  - Frames and enforced properties work together to allow incremental refinement
  - Relational linear temporal logic supports compositionality
  - Theory supports proofs of safety and progress properties
  - Fairness assumptions can be encoded using enforced properties
- Future work:
  - Real-time controllers - teleo-reactive programs, sampling logic
  - Tool support