Action Programming
In Rewriting Logic

LMU PST
Lenz Belzner
Action Programming

”The art and science of devising high-level control strategies for autonomous systems which employ a mental model of their environment and which reason about their actions as a means to achieve their goals.”

[Thielescher]
Action Programming

- Given (within the scope of this talk)
  - First-order state and dynamics representation
  - First-order, non-deterministic procedural program
    - Choice of actions & action arguments

- Reasoning about states & actions
  - Does a certain property hold in the current state?
  - Can an action be applied?
  - What changes in which way(s) due to execution?
Rewriting Logic

- Algebraic specification & term rewriting
  - Formal software specification
  - Program verification

- Formal computational semantics
  - Non-determinism
  - Concurrency

- Action Programming
  - Reasoning about state properties via matching
  - Reasoning about actions via rewrite rules
Matching & Rewriting

- Consider operation $\circ$ associative and commutative

- $a \circ b := \text{ext.} \ b \circ c \circ a$
  - because $[b \circ c \circ a] = [a \circ b \circ c]$

- $a \circ X := \text{ext.} \ b \circ c \circ a$
  - Two matches, with substitutions $\{X \leftarrow b, X \leftarrow c\}$

- Consider rewrite rule: $a \circ X \rightarrow X \circ X$
  - $a \circ b \circ c \rightarrow b \circ b \circ c$
  - $a \circ b \circ c \rightarrow b \circ c \circ c$

- Matching
  - Efficient, $O(\log n)$ for typical AC-terms
  - Allows for first-order (\& order-sorted) representation
Environmental Model

- **Fluents & Actions**
  - Fluents are environmental properties affected by actions
  - Fluents and actions as parametrizable sorts

- **States**
  - AC-terms of fluents and actions
  - Conjunction/Disjunction
    
    \[
    \text{truckIn(truck, rome)} \land \text{boxIn(box, rome)}
    \]

  - Also actions are allowed in state terms
    
    \[
    \text{truckIn(truck, rome)} \land \text{boxIn(box, rome)} \land \text{load(truck, box)}
    \]
Projection

- Does a particular state have a certain property?

- Generally: State **subsumption**
  - Abstract state *subsumes* concrete state
  - Related by substitution(s) of variables

- Considering free variables existentially quantified
  - $\text{boxIn}(B, C)$ *subsumes* $\text{boxIn}(\text{box, rome}) \land \text{boxIn}(\text{box', paris})$
  - $\text{boxIn}(\text{box, rome})$ *does not subsume* $\text{boxIn}(B, C)$
  - $\text{boxIn}(B, C) \land \text{boxIn}(B', C')$ *does not subsume* $\text{boxIn}(B'', C'')$
    (with unique name assumption)

- Subsumption in rewriting logic
  - **Matching with extension** (mod. axioms)
  - Abstract state *matches* concrete state
Rewriting & Actions

- Action context & effects as rewrite rules

\[ \sigma_{\text{precondition}} \land \sigma_{\text{affected}} \land \text{action(s)} \rightarrow \sigma_{\text{unchanged}} \land \sigma_{\text{effect}} \]

\[
\text{truckIn}(T, C) \land \text{boxIn}(B, C) \land \text{load}(T, B) \rightarrow \\
\text{truckIn}(T, C) \land \text{boxOn}(B, T)
\]

- Solution to the frame problem
Given
- Current state $\sigma$
- Program to execute $\pi$
- Deterministic action trace $\tau$

Rewrite current program
- $\pi \rightarrow \alpha_1 ; \pi_1 \# \alpha_2 ; \pi_2 \# \ldots$

Add action(s) to current state
- $\sigma \rightarrow \sigma \land \alpha_1$ or $\sigma \land \alpha_2$ or $\ldots$

Rewrite to new state(s) according to effect rules
- $\sigma \land \alpha_i \rightarrow \sigma_i$

Keep track of action trace
- $\tau \rightarrow \tau ; \alpha_i$
Non-Determinism

- Multiple rewrite laws for term $t$
  
  $t \rightarrow t'$
  $t \rightarrow t''$

- $t$ rewrites to $t'$ or $t''$

- Specification of non-deterministic action effects
  
  $\sigma \land \text{action(s)} \rightarrow \sigma'$
  $\sigma \land \text{action(s)} \rightarrow \sigma''$

- Application of rule(s) to overlapping matches
Concurrency

- Construct joint actions from parallel programs
  \[
  \alpha ; \pi \parallel \alpha' ; \pi' \rightarrow \alpha \land \alpha'; (\pi \parallel \pi')
  \]

- Unchanged specification of actions
  \[
  \text{truckIn}(T, C) \land \text{driveTo}(T, C') \rightarrow \text{truckIn}(T, C')
  \]

- Concurrent computation
  \[
  \text{truckIn}(t, rome) \land \text{truckIn}(t', paris) \land \\
  \text{driveTo}(t, madrid) \land \text{driveTo}(t', madrid) \rightarrow \\
  \text{truckIn}(t, madrid) \land \text{truckIn}(t', madrid)
  \]

- Application of rule(s) to multiple non-overlapping matches
Conclusion

- Reasoning about states
  - State subsumption
  - AC-terms and matching

- Reasoning about actions
  - Action dynamics as rewrite rules
  - Solution to the frame problem

- Non-determinism & concurrency

- *Procedural* language with *progressive* dynamics
  - Implementation in Maude
  - Order-sorted, polymorphic specification
  - Meta-programming facilities
  - Model checker
Implementation

http://www.pst.ifi.lmu.de/~belzner/action-programming/
Further Work

- Narrowing
  - Incomplete state knowledge

- Negation & Matching
  - Application of rewrite laws on states with negation?

- Ramification
  - Cyclic dependencies

- Qualification Problem
  - Unexpected action failure

- Relation to existing formalisms & languages
  - Unifying Action Calculus
  - Agent Logic Programs
Quantitative Uncertainty

- Decision-theoretic program trace evaluation
  - State transition rewards (e.g. as for MDPs)
  - Action effect probabilities
  - Extend states to a state-value tuple (e.g. State x Float)

- Action rules for extended states

\[(\sigma \land \alpha, v) \rightarrow (\sigma', (v + v_{\text{transition}}) \times p_{\text{transition}})\]
Quantitative Uncertainty

- \(((\sigma \land \alpha, v), \pi, \tau) \rightarrow \\
  ((\sigma', v'), \pi, \tau; \alpha) \text{ or} \\
  ((\sigma'', v''), \pi, \tau; \alpha) \text{ or} \\
  \ldots\)

- \(((\sigma', v'), \pi, \tau; \alpha) \text{ or } ((\sigma'', v''), \pi, \tau; \alpha) = \\
  ((\sigma' \lor \sigma'', v' + v''), \pi, \tau; \alpha)\)

- For \textit{trace value} computation rewrite \((\sigma_{\text{init}}, 0.0), \pi, \tau_{\text{empty}})\)
Rewriting & Matching

- **Rewrite rules**
  - lhs $\rightarrow$ rhs (if conditions)

- **Consider term t to be rewritten**
  - lhs matches subterm $t'$ of $t$ for substitution $\theta$
  - $\theta(t')$ will rewrite to $r$

- **t matches u if there is a substitution $\theta$ such that $\theta(t) = u$**

- **Matching *modulo axioms***
  - Matching on term equivalence classes

- **Matching *with extension***
  - Matching of subterms
Non-Deterministic Effects

\[ \sigma \land \alpha \rightarrow \sigma' \]
\[ \sigma \land \alpha \rightarrow \sigma'' \]
etc.

\[ \sigma \]
\[ \alpha \]
\[ \alpha \]
\[ \text{etc.} \]
\[ \sigma' \]
\[ \pi \]
\[ \sigma'' \]
\[ \pi \]
Projection
How to specify which alternatives to consider when?
Program Syntax

\[ \Pi ::= \text{nil} | \text{noop} | a(\bar{x}) | \Pi; \Pi | \Pi \# \Pi | \Pi \mid_{\text{det}} \Pi | \Pi \mid_{\text{nondet}} \Pi | \Pi \mid_{\text{true}} \Pi | \text{if } \Sigma \text{ then } \Pi \text{ else } \Pi \text{ end} \]

moveUp(robot) ;
moveLeft(robot) # moveRight(robot)
Sequence

\[ \alpha ; \pi \]

\[ \sigma \]

\[ \sigma' \]
Choice

\[ \alpha \neq \alpha'; \pi \]

\[ \sigma \]

\[ \sigma' \]

\[ \sigma'' \]
Concurrency
process

\text{process}(\pi, \sigma) \rightarrow \alpha ; \pi
\text{process}(\pi, \sigma) \rightarrow \alpha' ; \pi'
\text{etc.}
Semantics: \textit{process}

\[
\text{process} : \Pi \times \Sigma \rightarrow \Pi
\]

\[
\text{process}(\text{nil}, \sigma) = \text{noop};\text{nil}
\]

\[
\text{process}(a(\bar{x}), \sigma) = a(\bar{x});\text{nil}
\]

\[
\text{process}(\pi; \pi', \sigma) = \sigma_\alpha; \pi''; \pi
\]

\[
\text{if process}(\pi, \sigma) \rightarrow \sigma_\alpha; \pi''
\]

\[
\text{process}(\pi\#\pi', \sigma) \rightarrow \text{process}(\pi, \sigma)
\]

\[
\text{process}(\pi \parallel_{\text{det}} \pi', \sigma) \rightarrow \sigma_\alpha; (\pi' \parallel_{\text{det}} \pi'')
\]

\[
\text{if process}(\pi, \sigma) \rightarrow \sigma_\alpha; \pi''
\]

\[
\text{process}(\pi \parallel_{\text{ndet}} \pi', \sigma) \rightarrow \sigma_\alpha; (\pi' \parallel_{\text{ndet}} \pi'')
\]

\[
\text{if process}(\pi, \sigma) \rightarrow \sigma_\alpha; \pi''
\]

\[
\text{process}(\pi \parallel_{\text{true}} \pi', \sigma) \rightarrow \sigma_\alpha \land \sigma_\alpha'; (\pi'' \parallel_{\text{true}} \pi''''')
\]

\[
\text{if process}(\pi, \sigma) \rightarrow \sigma_\alpha; \pi''
\]

\[
\text{and process}(\pi', \sigma) \rightarrow \sigma_\alpha'; \pi'''
\]
How to reason about states resulting from action choices?
State Representation

$$
\Sigma := \varepsilon \mid \text{noop} \mid f(\vec{x}) \mid a(\vec{x}) \mid \Sigma \land \Sigma \mid \Sigma \lor \Sigma
$$

pos(robot,1,1) \land \text{moveUp}(robot)

pos(R, X, Y) \land \text{moveUp}(R)
holds in
State subsumption

- \( \text{pos}(R, X, Y) \text{ holds in } \text{pos}(R', X', Y') \)
- \( \text{pos}(R, X, Y) \text{ holds in } \text{pos(\text{wall-e}, 1, 2)} \)
- \( \text{pos(\text{wall-e}, 1, 2) not holds in } \text{pos}(R, X, Y) \)

- \( \text{pos}(R, X, Y) \text{ holds in } \text{pos}(R', X', Y') \land \text{pos}(R'', X'', Y'') \)
- \( \text{pos}(R, X, Y) \land \text{pos}(R', X', Y') \text{ not holds in } \text{pos}(R'', X'', Y'') \)

- More general state \( \text{holds in} \) less general state
  - for some substitution of variables
  - \( \rightarrow \) Semantics: \textit{matching with extension}
- Free variables are considered \textit{existentially quantified}
Branching
Semantics: \textit{process} (branching)

\[
\text{process}(\text{if } \sigma_{\text{cond}} \text{ then } \pi \text{ else } \pi' \text{ end}, \sigma) \rightarrow \text{process}(\pi[\bar{x}/\text{unifier}], \sigma) \\
\text{if } \text{unify}(\sigma_{\text{cond}}, \sigma) \rightarrow \text{unifier} \\
\text{process}(\pi', \sigma) \\
\text{if } \text{unify}(\sigma_{\text{cond}}, \sigma) \rightarrow \bot
\]
How to represent domain dynamics?
Dynamics & Effects

- **Tense** current state term with actions according to agent programs
  - State: \( \text{pos} (\text{robot}, 1, 1) \)
  - Program: \( \text{moveUp} (\text{robot}) \)
  - Tensed state: \( \text{pos} (\text{robot}, 1, 1) \land \text{moveUp} (\text{robot}) \)

- **Relax** tensed term w.r.t. domain dynamics (rewrite laws)
  - Effect law: \( \text{pos} (R, X, Y) \land \text{moveUp} (R) \rightarrow \text{pos} (R, X, Y + 1) \)
  - Relaxed state: \( \text{pos} (\text{robot}, 1, 2) \)
Actions (Domain Dynamics)

- State: $\sigma$
- Program: $\pi$
- $\alpha_1$
- $\alpha_2$
- etc.

- State: $\sigma \land \alpha_1$
  - Program: $\pi_1$

- State: $\sigma \land \alpha_2$
  - Program: $\pi_2$

- $\alpha_1; \pi_1 \# \alpha_2; \pi_2 \# ...$
Actions (Domain Dynamics)

State: $\sigma$
Program: $\pi$

$\alpha_1$
$\alpha_2$

State: $\sigma \land \alpha_1$
Program: $\pi_1$

State: $\sigma \land \alpha_2$
Program: $\pi_2$

$\sigma \land \alpha_1 \Rightarrow \sigma_1$
$\sigma \land \alpha_2 \Rightarrow \sigma_2$

etc.
Actions (Domain Dynamics)
Actions

\( \sigma_{prec\cup affected} \land \sigma_\alpha \rightarrow \sigma_{unchanged\cup effect} \) if Conditions

\( \text{truckIn}(T, C) \land \text{boxIn}(B, C) \land \text{load}(T, B) \rightarrow \text{truckIn}(T, C) \land \text{boxOn}(B, T) \)
Frame Problem

position(P) and energy(0.9) and S

moveTo(P') charge()

position(P') and energy(0.8) and S

position(P) and energy(1.0) and S

... ...

... ...

... ...
How to generate plans?
Putting it together: exec

\[ \text{process}(\pi, \sigma) \rightarrow \alpha ; \pi \]

\[ \text{process}(\pi, \sigma) \rightarrow \alpha' ; \pi' \]

\[ \text{etc.} \]

\[ \sigma \land \alpha \rightarrow \sigma' \]

\[ \sigma \land \alpha' \rightarrow \sigma'' \]
Plans

\[ \text{exec}(\sigma, \pi, \tau) \Rightarrow \]
\[ \sigma_{\text{final}} ; \tau_{\text{final}} \]
Semantics: \( \text{exec} \)

\[
\text{exec} : \Sigma \times \Pi \times \Pi \to \Sigma \times \Pi
\]

- \( \text{exec}(\sigma, \text{nil}, \tau) \to \sigma \times \tau \)
- \( \text{exec}(\sigma \lor \sigma', \pi, \tau) \to \text{exec}(\sigma, \pi, \tau) \) and \( \pi \neq \text{nil} \)
- \( \text{exec}(\sigma, \pi, \tau) \to \text{exec}(\sigma_{\text{relaxed}}, \pi', \tau; \sigma_{\alpha}) \) if \( \text{process}(\pi, \sigma) \to \sigma_{\alpha}; \pi' \) and \( \sigma \land \sigma_{\alpha} \to \sigma_{\text{relaxed}} \) and \( \sigma' \land \sigma'' := \sigma \) and \( \pi \neq \text{nil} \)
Summing it up
Summing It Up

- How to specify which alternatives to consider when?
  - **Non-deterministic programs**
- How to reason about states resulting from action choices?
  - **Matching & state subsumption**
- How to represent domain dynamics?
  - **Rewrite laws**
- How to generate plans?
  - **Traces in TS**
Further Work

- Formal integration with action formalisms
  - SC, FC, EC, ALP
- MDPs, RRL & SDP
  - Rates, Markov Chains?
- Exogenous events, sensing actions, decision theory, invariants
- Explicit negation & constrained states
- Imperative extension (OO, state-based programming)
- Maude (LTL) model checking
- Multi-agent settings & game theory
- Process Algebra? Synchronization? Tool Support?
  - Unconsumed actions
  - Petri Nets?
Are there LTS with variables in the labels corresponding to state variables? DLTS?

Is there a mapping from stochastic process algebra with rates to MDPs and their solutions?

12 Minutes?
Appendix
Rewrite Logic

- Rewrite theory: \((\Sigma, E \cup A, \phi, R)\)

- \(\Sigma\): Set of sorts and operations
- \(E \cup A\): Set of (conditional) equations and axioms (for operations)
  - \(X \circ e = X\) if \(X \neq 0\)
  - associativity, commutativity, idempotency, identity
- \(\phi\): Set of frozen arguments
- \(R\): Set of rewrite laws
  - [label] \(t \rightarrow t'\) if Conditions
Non-Determinism (Effects)

- position(P) and energy(0.9)
  - moveTo(P')
  - moveTo(P')
    - position(P') and energy(0.8)
    - position(P) and energy(0.7)
Truly Concurrent Actions

\[ \text{process} : \Pi \times \Sigma \rightarrow \Pi \]

\[
\begin{align*}
\text{process}(\text{nil}, \sigma) &= \text{noop};\text{nil} \\
\text{process}(a(\bar{x}), \sigma) &= a(\bar{x});\text{nil} \\
\text{process}(\pi;\pi', \sigma) &= \sigma\alpha;\pi'';\pi \\
\text{if} &\quad \text{process}(\pi, \sigma) \rightarrow \sigma\alpha;\pi'' \\
\text{process}(\pi\#\pi', \sigma) &\rightarrow \text{process}(\pi, \sigma) \\
\text{process}(\pi \parallel_{\text{det}} \pi', \sigma) &\rightarrow \sigma\alpha; (\pi' \parallel_{\text{det}} \pi'') \\
\text{if} &\quad \text{process}(\pi, \sigma) \rightarrow \sigma\alpha;\pi'' \\
\text{process}(\pi \parallel_{\text{ndet}} \pi', \sigma) &\rightarrow \sigma\alpha; (\pi' \parallel_{\text{ndet}} \pi'') \\
\text{if} &\quad \text{process}(\pi, \sigma) \rightarrow \sigma\alpha;\pi'' \\
\text{process}(\pi \parallel_{\text{true}} \pi', \sigma) &\rightarrow \sigma\alpha \land \sigma\alpha'; (\pi'' \parallel_{\text{true}} \pi''') \\
\text{if} &\quad \text{process}(\pi, \sigma) \rightarrow \sigma\alpha;\pi'' \\
\text{and} &\quad \text{process}(\pi', \sigma) \rightarrow \sigma\alpha';\pi'''
\end{align*}
\]
Truly Concurrent Actions

\[ T, T' : Truck, C, C', C'' : City \]

\[ truckIn(T, C) \land truckIn(T', C'') \land driveTo(T, C') \land driveTo(T', C') \rightarrow truckIn(T, C') \land truckIn(T', C') \land trafficJam(C') \]
Remarks

- Clarify TS <-> LTS
- Time model(s)
- Reduce process, encoding part
- Emphasize separation of behaviour and model
- Emphasize plan generation and goal verification
- CTL?