TU Clausthal
Clausthal University of Technology

Towards pragmatics of rule-based agent programming language(s)

(on-going work)

Peter Novák

Clausthal University of Technology, Germany

September 4th, 2008
Dagstuhl Seminar 08361 - ProMAS

Problem: *pragmatics of programming with APLs*

***generic** programming language for cognitive agents*

- mixing heterogeneous KRs: not fixed agent architecture
- non-determinism/reactivity: interleaving behaviours
    - driver apps: cognitive (simulated) robotics

specification $\phi \rightsquigarrow$ program $\mathcal{P}$

ϕ — decomposition → $\mathcal{P}$

$\mathcal{P}$ vs. ϕ — verification → ϕ

Support of design process by code templates/idioms/design patterns...

Problem: *pragmatics of programming with APLs*

***generic** programming language for cognitive agents*

- mixing heterogeneous KRs: not fixed agent architecture
- non-determinism/reactivity: interleaving behaviours
  - driver apps: cognitive (simulated) robotics

specification $\phi \rightsquigarrow$ program $\mathcal{P}$

ϕ ⇣ **decomposition** ⇣ $\mathcal{P}$

$\mathcal{P}$ **vs.** ϕ ⇡ **verification** ⇡ ϕ

Support of design process by code templates/idioms/design patterns...

The way to go...

High level code structures have to:

- clearly *characterize* the encapsulated code
- allow further combination $\rightsquigarrow$ *compositionality*

} formally capture meaning of code structures!

Thesis:

Mixture of Dynamic Logic + Temporal Logic allows for capturing/extraction characterization of implemented code.

Agenda:

1. verification step
2. refinement with code templates (sketch)

The way to go...

High level code structures have to:

- clearly *characterize* the encapsulated code
- allow further combination ⇝ *compositionality*

} formally capture meaning of code structures!

Thesis:

Mixture of Dynamic Logic + Temporal Logic allows for capturing/extraction characterization of implemented code.

Agenda:

1. verification step
2. refinement with code templates (sketch)

Behavioural State Machines/Jazzyk

Behavioural State Machines/Jazzyk

A *lightweight* programming framework with clear separation between *knowledge representation* and agent's *behaviours*.

reasoning vs. computation model

BSM agent: $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$

KR module $\mathcal{M} = (\mathcal{S}, \mathcal{L}, \mathcal{Q}, \mathcal{U})$

- $\mathcal{S}$ - a set of states
- $\mathcal{L}$ - a KR language,
- $\mathcal{Q}$ - a set of query operators $\models: \mathcal{S} \times \mathcal{L} \rightarrow \{\top, \bot\}$,
- $\mathcal{U}$ - set of update operators $\oslash : \mathcal{S} \times \mathcal{L} \rightarrow \mathcal{S}$.

Behavioural State Machines (cont.)

mental state transformer:

$$\models_i \varphi \longrightarrow \oslash_j \psi$$

when query$_i$ **module**$_i$ [{ φ }] **then** update$_j$ **module**$_j$ [{ ψ }]

$\tau_1 | \tau_2$ non-deterministic choice, $\tau_1 \circ \tau_2$ sequence

Jazzyk BSM semantics (operational view)

A sequence $\sigma_1, \ldots, \sigma_i, \ldots$, s.t. $\sigma_i \rightarrow \sigma_{i+1}$, is a trace of BSM.
An agent system (BSM), is characterized by a set of all traces.

transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ induced by updates $\oplus\psi$ yielded by the agent program $\mathcal{P}$

$\leadsto$ (Novák, Dix @ AAMAS'06, ..., ProMAS'08)

Behavioural State Machines (cont.)

mental state transformer:

$$\models_i \varphi \longrightarrow \oslash_j \psi$$

when query$_i$ **module**$_i$ [{ φ }] **then** update$_j$ **module**$_j$ [{ ψ }]

$\tau_1 | \tau_2$ non-deterministic choice, $\tau_1 \circ \tau_2$ sequence

Jazzyk BSM semantics (operational view)

A sequence $\sigma_1, \ldots, \sigma_i, \ldots$, s.t. $\sigma_i \rightarrow \sigma_{i+1}$, is a trace of BSM.
An agent system (BSM), is characterized by a set of all traces.

transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ induced by updates $\oplus\psi$ yielded by the agent program $\mathcal{P}$

$\leadsto$ (Novák, Dix @ AAMAS'06, ..., ProMAS'08)

```
/* When the searched item is found, pick it */
when desires_G [{ task(pick(X)) }] then {
    /* PICK */
    when believes_B [{ see(X) }] then {
        when believes_B [{ dir(X,'ahead'), dist(X,Dist) }] then act_E [{ move forward Dist }] |
        when believes_B [{ dir(X, Angle) }] then act_E [{ turn Angle }]
    } . . . |
} |
/* Goal adoption */
when believes_B [{needs(X)}] then add_G [{task(pick(X))}] |
/* Drop the goal */
when desires_G [{ task(pick(X)) }] and believes_B [{holds(X)}] then remove_G [{task(pick(X))}] |

/* When endangered, run away */
when desires_G [{ maintain(safety) }] and believes_B [{ threatened }] then {
    /* RUN_AWAY */
    when believes_B [{ random(Angle) }] then {
        act_E [{ turn Angle }] ∘
        act_E [{ move forward 10 }]
    } |
    . . .
}
```

```
/* When the searched item is found, pick it */
ACHIEVE('task(pick(X))', 'needs(X)', 'holds(X)', PICK)
 |
/* When endangered, run away */
MAINTAIN('maintain(safety)', 'threatened', RUN_AWAY)
```

⇝ code adapted from (Novák, Köster @ CogRob'08)

```
/* When the searched item is found, pick it */
when desires_G [{ task(pick(X)) }] then {
    /* PICK */
    when believes_B [{ see(X) }] then {
        when believes_B [{ dir(X,'ahead'), dist(X,Dist) }] then act_E [{ move forward Dist }] |
        when believes_B [{ dir(X, Angle) }] then act_E [{ turn Angle }]
    } . . . |
} |
/* Goal adoption */
when believes_B [{needs(X)}] then add_G [{task(pick(X))}] |
/* Drop the goal */
when desires_G [{ task(pick(X)) }] and believes_B [{holds(X)}] then remove_G [{task(pick(X))}] |

/* When endangered, run away */
when desires_G [{ maintain(safety) }] and believes_B [{ threatened }] then {
    /* RUN_AWAY */
    when believes_B [{ random(Angle) }] then {
        act_E [{ turn Angle }] ∘
        act_E [{ move forward 10 }]
    } |
    . . .
}
```

```
/* When the searched item is found, pick it */
ACHIEVE('task(pick(X))', 'needs(X)', 'holds(X)', PICK)
  |
/* When endangered, run away */
MAINTAIN('maintain(safety)', 'threatened', RUN_AWAY)
```

⇝ code adapted from (Novák, Köster @ CogRob'08)

DLTL Syntax

$\rightsquigarrow$ (Henriksen, Thiangarajan @ Ann. Pure Appl. Logic'99)

Δ - operations, Σ - atomic propositions

$Plain(\Sigma)$ - propositional formulae: φ, $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$

$DLTL(\Sigma, \Delta)$

- $Plain(\Sigma) \subseteq DLTL(\Sigma, \Delta)$
- $\varphi\mathcal{U}^{\pi}\psi \in DLTL(\Sigma, \Delta)$ $\qquad \varphi, \psi \in DLTL(\Sigma, \Delta)$, $\pi \in Prg(\Sigma, \Delta)$

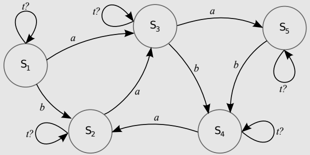
$Prg(\Sigma, \Delta)$

- $\Delta \cup \{\varepsilon\} \subseteq Prg(\Sigma, \Delta)$ (atomic)
- if $\varphi \in Plain(\Sigma)$, then $\varphi? \in Prg(\Sigma, \Delta)$ (test)
- if $\tau_1, \tau_2 \in Prg(\Sigma, \Delta)$, then $\tau_1 | \tau_2$, $\tau_1 \circ \tau_2 \in Prg(\Sigma, \Delta)$ (compound)
- if $\tau \in Prg(\Sigma, \Delta)$, then also $\tau^* \in Prg(\Sigma, \Delta)$ (iteration)

Mapping to words: $|| \cdot || : Prg(\Sigma, \Delta) \rightarrow 2^{\Delta^*}$

Semantics

labeled transition system $K = (S, R, \Delta, \Delta^?, \Phi_S, \Sigma)$



path , $Lbl(\varsigma) = abt?a$

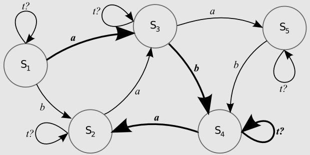
Semantics: $\models: \mathcal{K} \times S \times DLTL(\Sigma, \Delta) \rightarrow \{\top, \bot\}$

$K, \sigma \models \varphi \mathcal{U}^\pi \psi$:

- $\exists \varsigma = \varsigma_1 \bullet \varsigma_2 \subseteq K$, s.t. $head(\varsigma) = \sigma$, $Lbl(\varsigma) \in ||\pi||$
  - $last(\varsigma_1) = \sigma' \implies K, \sigma' \models \psi$.
  - $\forall \sigma'' \subseteq \varsigma_1 \implies K, \sigma'' \models \varphi$.

Semantics

labeled transition system $K = (S, R, \Delta, \Delta^?, \Phi_S, \Sigma)$



path $\varsigma = s_1 \xrightarrow{a} s_3 \xrightarrow{b} s_4 \xrightarrow{t?} s_4 \xrightarrow{a} s_2$, $Lbl(\varsigma) = abt?a$

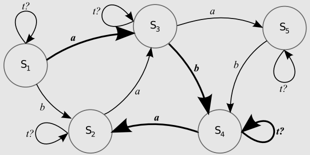
Semantics: $\models : \mathcal{K} \times S \times DLTL(\Sigma, \Delta) \to \{\top, \bot\}$

$K, \sigma \models \varphi \mathcal{U}^\pi \psi$:

- $\exists \varsigma = \varsigma_1 \bullet \varsigma_2 \subseteq K$, s.t. $head(\varsigma) = \sigma$, $Lbl(\varsigma) \in ||\pi||$
  - $last(\varsigma_1) = \sigma' \implies K, \sigma' \models \psi$.
  - $\forall \sigma'' \subseteq \varsigma_1 \implies K, \sigma'' \models \varphi$.

Semantics

labeled transition system $K = (S, R, \Delta, \Delta^?, \Phi_S, \Sigma)$



path $\varsigma = \underbrace{s_1 \xrightarrow{a} s_3 \xrightarrow{b} s_4}_{\varsigma_1} \underbrace{\xrightarrow{t?} s_4 \xrightarrow{a} s_2}_{\varsigma_2}$, $Lbl(\varsigma) = abt?a$

Semantics: $\models: \mathcal{K} \times S \times DLTL(\Sigma, \Delta) \to \{\top, \bot\}$

$K, \sigma \models \varphi \mathcal{U}^\pi \psi$:

- $\exists \varsigma = \varsigma_1 \bullet \varsigma_2 \subseteq K$, s.t. $head(\varsigma) = \sigma$, $Lbl(\varsigma) \in ||\pi||$
  - $last(\varsigma_1) = \sigma' \Longrightarrow K, \sigma' \models \psi$.
  - $\forall \sigma'' \subseteq \varsigma_1 \Longrightarrow K, \sigma'' \models \varphi$.

Derived DLTL modalities

- $\langle\pi\rangle\varphi \overset{def}{\Longleftrightarrow} \top\mathcal{U}^{\pi}\varphi$
- $[\pi]\varphi \overset{def}{\Longleftrightarrow} \neg\langle\pi\rangle\neg\varphi$
- $\bigcirc\varphi \overset{def}{\Longleftrightarrow} \bigvee_{p\in\Delta}\langle p\rangle\varphi$
- $\varphi\mathcal{U}\psi \overset{def}{\Longleftrightarrow} \varphi\mathcal{U}^{\pi_\Delta^*}\psi$
- $\Diamond\varphi \overset{def}{\Longleftrightarrow} \top\mathcal{U}\varphi$
- $\Box\varphi \overset{def}{\Longleftrightarrow} \neg\Diamond\neg\varphi$

$$\pi_\Delta = p_1|p_2|\cdots|p_n,\ \Delta = \{p_1,\ldots,p_n\}$$

LTL $\subset$ DLTL

Software engineering problem revisited

$\phi \in LTL(\Sigma)$ $\quad\quad$ $[\mathcal{P}]\psi \in DLTL(\Sigma, \Delta)$

refinement $[?]\phi$ $\quad\quad$ $[\mathcal{P}]?$ **characterization**

$\mathcal{P} \in BSM$ $\quad\quad$ $\mathcal{P} \in BSM$

DLTL can help us in both directions!

Software engineering problem revisited

$\phi \in LTL(\Sigma)$

refinement $[?]\phi$

$\mathcal{P} \in BSM$

$[\mathcal{P}]\psi \in DLTL(\Sigma, \Delta)$

$[\mathcal{P}]?$ **characterization**

$\mathcal{P} \in BSM$

DLTL can help us in both directions!

Annotated BSM

Annotated Behavioural State Machine

... is an extension of a BSM $\mathcal{A}$ with annotated primitive query and update formulae: (Σ atomic propositions, Δ atomic operations)

- $\Phi_S : \mathcal{S}_1 \times \cdots \times \mathcal{S}_n \rightarrow 2^{\Sigma}$ state labeling function
- $\Phi_{\models} : \bigcup_{i=1}^{n}(\mathcal{Q}_1 \times \mathcal{L}_1) \rightarrow Plain(\Sigma)$ query annotation, i.e. $\models \varphi \mapsto \phi?$
- $\Phi_{\oslash} : \bigcup_{i=1}^{n}(\mathcal{U}_1 \times \mathcal{L}_1) \rightarrow \Delta$ update annotation, i.e. $\oslash\psi \mapsto a$
- $STRIPS : \Delta \rightarrow Plain(\Sigma)$ action characterization, i.e. $a \mapsto \phi_{add} \wedge \phi_{del}$

Translation: heterogeneous KRs $\rightsquigarrow$ single KR language!

$\rightsquigarrow$ similar to (Dastani, Hindriks, Tinnemeyer, Novák @ DALT'08)

Capturing the program meaning

characterization extraction

A BSM program $\mathcal{P}$ is characterized by a DLTL formula $\mathfrak{T}(\mathcal{P})$:

1. mst's:
   - $\mathfrak{T}(\mathbf{skip}) = [\varepsilon] \bigcirc \top$
   - $\mathfrak{T}(\oslash\psi) = [a] \bigcirc STRIPS(a)$ $\quad a = \Phi_{\oslash}(\oslash, \psi)$
   - $\mathfrak{T}(\tau_1 | \tau_2) = [\pi_{\tau_1} | \pi_{\tau_2}]\varphi_{\tau_1} \vee \varphi_{\tau_2}$ $\quad \mathfrak{T}(\tau_i) = [\pi_i]\varphi_i$
   - $\mathfrak{T}(\tau_1 \circ \tau_2) = [\pi_{\tau_1} \circ \pi_{\tau_2}]\varphi_1 \mathcal{U} \varphi_2$ $\quad \mathfrak{T}(\tau_i) = [\pi_i]\varphi_i$
   - $\mathfrak{T}(\phi \longrightarrow \tau) = [\psi_\phi? \circ \pi_\tau]\psi_\phi \mathcal{U} \varphi_\tau$ $\quad \mathfrak{T}(\tau) = [\pi_\tau]\varphi_\tau, \mathfrak{T}(\phi) = \psi_\phi?$

reasoning about incomplete annotations:

- choice of an appropriate level of abstraction
- choice of an aspect of the agent program to verify

Capturing the program meaning

characterization extraction

A BSM program $\mathcal{P}$ is characterized by a DLTL formula $\mathfrak{T}(\mathcal{P})$:

1. mst's:
   - $\mathfrak{T}(\mathbf{skip}) = [\varepsilon] \bigcirc \top$
   - $\mathfrak{T}(\oslash\psi) = [a] \bigcirc STRIPS(a)$ $\quad a = \Phi_{\oslash}(\oslash, \psi)$
   - $\mathfrak{T}(\tau_1|\tau_2) = [\pi_{\tau_1}|\pi_{\tau_2}]\varphi_{\tau_1} \vee \varphi_{\tau_2}$ $\quad \mathfrak{T}(\tau_i) = [\pi_i]\varphi_i$
   - $\mathfrak{T}(\tau_1 \circ \tau_2) = [\pi_{\tau_1} \circ \pi_{\tau_2}]\varphi_1 \mathcal{U} \varphi_2$ $\quad \mathfrak{T}(\tau_i) = [\pi_i]\varphi_i$
   - $\mathfrak{T}(\phi \longrightarrow \tau) = [\psi_\phi ? \circ \pi_\tau]\psi_\phi \mathcal{U} \varphi_\tau$ $\quad \mathfrak{T}(\tau) = [\pi_\tau]\varphi_\tau$, $\mathfrak{T}(\phi) = \psi_\phi ?$

reasoning about incomplete annotations:

- choice of an appropriate level of abstraction
- choice of an aspect of the agent program to verify

Software engineering problem again

$\phi \in LTL(\Sigma)$ — refinement — $[?]\phi$ — $\mathcal{P}$

$\mathfrak{T}(\mathcal{P}) \equiv [\tau]\psi$ ← characterization — $\mathcal{P}$

verification: $[\tau]\psi \stackrel{?}{\Longrightarrow} [\tau^*]\phi$ ψ characterization, ϕ specification

- APLs deliberation cycle: *program iteration*
- model checking(?)
- theorem prover?

decomposition: serie of refining steps down to atomic operations corresponding to primitive mst's

Software engineering problem again

$\phi \in LTL(\Sigma)$ — refinement $[?]\phi$ — $\mathcal{P}$

$\mathfrak{T}(\mathcal{P}) \equiv [\tau]\psi$ — characterization — $\mathcal{P}$

verification: $[\tau]\psi \overset{?}{\Longrightarrow} [\tau^*]\phi$ ψ characterization, ϕ specification

- APLs deliberation cycle: *program iteration*
- model checking(?)
- theorem prover?

decomposition: serie of refining steps down to atomic operations corresponding to primitive mst's

Decomposition: sketch

gradual ***refinement*** of the specification

- verification $\rightsquigarrow$ compositional semantics for compound structures

Example

1. $S_1 \equiv \varphi$
2. $S_2 \equiv \phi_1 \wedge \phi_2 \vee \phi_3$ and $\phi_1 \wedge \phi_2 \vee \phi_3 \Rightarrow \varphi$
3. $S_3 \equiv [ACHIEVE(\pi_1)]\phi_1 \wedge \ldots \vee [MAINTAIN(\pi_3)]\phi_3$
4. $\ldots$
5. $S_5 \equiv [\mathcal{P}](\phi_1 \wedge \ldots \vee \phi_3)$ and $\mathcal{P} \leftrightarrow \pi_1, \ldots, \pi_3$

$S_5 \Rightarrow S_4 \Rightarrow S_3 \Rightarrow S_2 \Rightarrow S_1 \equiv \varphi$

Intuition:

$[\tau^*]\Diamond\varphi$: τ implements ***achievement*** of φ — **ACHIEVE('φ',...)**

$[\tau^*]\Box\varphi$: τ implements ***maintenance*** of φ — **MAINTAIN('φ',...)**

Decomposition: sketch

gradual ***refinement*** of the specification

- verification $\rightsquigarrow$ compositional semantics for compound structures

Example

1. $S_1 \equiv \varphi$
2. $S_2 \equiv \phi_1 \wedge \phi_2 \vee \phi_3$ and $\phi_1 \wedge \phi_2 \vee \phi_3 \Rightarrow \varphi$
3. $S_3 \equiv [ACHIEVE(\pi_1)]\phi_1 \wedge \ldots \vee [MAINTAIN(\pi_3)]\phi_3$
4. $\ldots$
5. $S_5 \equiv [\mathcal{P}](\phi_1 \wedge \ldots \vee \phi_3)$ and $\mathcal{P} \leftrightarrow \pi_1, \ldots, \pi_3$

$$S_5 \Rightarrow S_4 \Rightarrow S_3 \Rightarrow S_2 \Rightarrow S_1 \equiv \varphi$$

Intuition:

$[\tau^*]\Diamond\varphi$: τ implements ***achievement*** of φ — **ACHIEVE('φ',...)**

$[\tau^*]\Box\varphi$: τ implements ***maintenance*** of φ — **MAINTAIN('φ',...)**

Conclusion

DL + *TL can provide insight into development of
high level code structures with clear semantics

- from specification to implementation: *creative process*
  - $\leadsto$ prefabricated code structures, patterns, templates

Library of agent-oriented idioms:

- various types of goals/commitment strategies
- control cycle: models of mixing behaviours:
  - (sense $\circ$ deliberate $\circ$ act) vs. (sense | deliberate | act) etc.

Practical experience $\leadsto$ structuring of larger code bases

Related work:

- *Jason*: code patterns
- *GOAL*: modules(?)
- *Abstract State Machines*: refinement

Thank you for your attention.

http://jazzyk.sourceforge.net/