Jazzyk: agents with heterogeneous knowledge representations

(programming language overview)

Peter Novák

Clausthal University of Technology, Germany

May 13th, 2008
ProMAS 2008, Estoril, Portugal

Programming cognitive agents

Different programming languages are suitable for different knowledge representation tasks.

Heterogeneous knowledge bases!

Focus on encoding agent's behaviours.

Behavioural State Machines

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

BSM framework provides:

- clear *semantics:* Gurevich's Abstract State Machines
- *modularity:* KR, source code
- easy *integration* with external/legacy systems

Programming cognitive agents

Different programming languages are suitable for different knowledge representation tasks.

Heterogeneous knowledge bases!

Focus on encoding agent's behaviours.

Behavioural State Machines

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

BSM framework provides:

- clear *semantics:* Gurevich's Abstract State Machines
- *modularity:* KR, source code
- easy *integration* with external/legacy systems

Programming cognitive agents

Different programming languages are suitable for different knowledge representation tasks.

Heterogeneous knowledge bases!

Focus on encoding agent's behaviours.

Behavioural State Machines

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

BSM framework provides:

- clear *semantics:* Gurevich's Abstract State Machines
- *modularity:* KR, source code
- easy *integration* with external/legacy systems

Jazzyk 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 $\models: \mathcal{S} \times \mathcal{L} \to \{\top, \bot\}$,
- $\mathcal{U}$ - set of update operators $\oplus : \mathcal{S} \times \mathcal{L} \to \mathcal{S}$.

mental state transformer τ: $\models_i \varphi \longrightarrow \oplus_j \psi$

```
when query$_i$ module$_i$ [{ $\varphi$ }] then update$_j$ module$_j$ [{ $\psi$ }]

when query$_i$ module$_i$ [{...}] and query$_j$ module$_j$ [{...}] then {
    when query$_k$ module$_k$ [{...}] then {
        ...
    } ;
    update$_l$ module$_l$ [{...}]
}
```

Jazzyk 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 $\oplus : \mathcal{S} \times \mathcal{L} \rightarrow \mathcal{S}$.

mental state transformer τ: $\models_i \varphi \longrightarrow \oplus_j \psi$

```
when query_i module_i [{ φ }] then update_j module_j [{ ψ }]

when query_i module_i [{...}] and query_j module_j [{...}] then {
     when query_k module_k [{...}] then {
          ...
     } ;
     update_l module_l [{...}]
}
```

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

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 $\oplus : \mathcal{S} \times \mathcal{L} \rightarrow \mathcal{S}$.

mental state transformer τ: $\models_i \varphi \longrightarrow \oplus_j \psi$

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

```
when query$_i$ module$_i$ [{...}] and query$_j$ module$_j$ [{...}] then {
    when query$_k$ module$_k$ [{...}] then {
        ...
    } ;
    update$_l$ module$_l$ [{...}]
}
```

Jazzyk 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 $\oplus : \mathcal{S} \times \mathcal{L} \rightarrow \mathcal{S}$.

mental state transformer τ: $\models_i \varphi \longrightarrow \oplus_j \psi$

```
when query$_i$ module$_i$ [{ $\varphi$ }] then update$_j$ module$_j$ [{ $\psi$ }]

when query$_i$ module$_i$ [{...}] and query$_j$ module$_j$ [{...}] then {
    when query$_k$ module$_k$ [{...}] then {
        ...
    } ;
    update$_l$ module$_l$ [{...}]
}
```

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

transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ induced by updates $\oplus\psi$

$$yields(\tau, \sigma, \theta, \rho)$$

$$\frac{\top}{yields(\textbf{skip},\sigma,\textbf{skip})} \qquad \frac{\top}{yields(\textbf{update}_{\oplus_i}\ \textbf{module}_i\ \psi,\sigma,\oplus_i\psi)} \qquad \frac{yields(\tau,\sigma,\rho)}{yields(\{\tau\},\sigma,\rho)}$$

$$\frac{yields(\tau,\sigma,\rho),\ \sigma\models_i\phi}{yields(\textbf{when query}_{\models_i}\ \textbf{module}_i\ \phi\ \textbf{then}\ \tau,\sigma,\rho)} \qquad \frac{yields(\tau,\sigma,\rho),\ \sigma\not\models_i\phi}{yields(\textbf{...},\sigma,\textbf{skip})}$$

$$\frac{yields(\tau_1,\sigma,\rho_1),\ yields(\tau_2,\sigma,\rho_2)}{yields(\tau_1;\tau_2,\sigma,\rho_1)\quad yields(\tau_1;\tau_2,\sigma,\rho_2)}$$

$$\frac{yields(\tau_1,\sigma,\rho_1),\ yields(\tau_2,\sigma \bigoplus \rho_1,\rho_2)}{yields(\tau_1,\tau_2,\sigma,\rho_1\bullet\rho_2)}$$

mst semantics: $\nu(\tau) = \{\rho | \exists\theta : yield(\tau, \sigma, \theta, \rho)\}$

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

transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ **induced by updates** $\oplus\psi$

$$yields(\tau, \sigma, \theta, \rho)$$

$$\frac{\top}{yields(\textbf{skip},\sigma,\textbf{skip})} \qquad \frac{\top}{yields(\textbf{update}_{\oplus_i}\ \textbf{module}_i\ \psi,\sigma,\oplus_i\psi)} \qquad \frac{yields(\tau,\sigma,\rho)}{yields(\{\tau\},\sigma,\rho)}$$

$$\frac{yields(\tau,\sigma,\rho),\ \sigma\models_i\phi}{yields(\textbf{when query}_{\models_i}\ \textbf{module}_i\ \phi\ \textbf{then}\ \tau,\sigma,\rho)} \qquad \frac{yields(\tau,\sigma,\rho),\ \sigma\not\models_i\phi}{yields(\textbf{...},\sigma,\textbf{skip})}$$

$$\frac{yields(\tau_1,\sigma,\rho_1),\ yields(\tau_2,\sigma,\rho_2)}{yields(\tau_1;\tau_2,\sigma,\rho_1) \quad yields(\tau_1;\tau_2,\sigma,\rho_2)}$$

$$\frac{yields(\tau_1,\sigma,\rho_1),\ yields(\tau_2,\sigma \bigoplus \rho_1,\rho_2)}{yields(\tau_1,\tau_2,\sigma,\rho_1 \bullet \rho_2)}$$

mst semantics: $\nu(\tau) = \{\rho | \exists\theta : yield(\tau, \sigma, \theta, \rho)\}$

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

transition system over states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ induced by updates $\oplus\psi$

$$yields(\tau, \sigma, \theta, \rho)$$

$$\frac{\top}{yields(\textbf{skip},\sigma,\textbf{skip})} \qquad \frac{\top}{yields(\textbf{update}_{\oplus_i}\ \textbf{module}_i\ \psi,\sigma,\oplus_i\psi)} \qquad \frac{yields(\tau,\sigma,\rho)}{yields(\{\tau\},\sigma,\rho)}$$

$$\frac{yields(\tau,\sigma,\rho),\ \sigma\models_i\phi}{yields(\textbf{when query}_{\models_i}\ \textbf{module}_i\ \phi\ \textbf{then}\ \tau,\sigma,\rho)} \qquad \frac{yields(\tau,\sigma,\rho),\ \sigma\not\models_i\phi}{yields(\textbf{...},\sigma,\textbf{skip})}$$

$$\frac{yields(\tau_1,\sigma,\rho_1),\ yields(\tau_2,\sigma,\rho_2)}{yields(\tau_1;\tau_2,\sigma,\rho_1)\quad yields(\tau_1;\tau_2,\sigma,\rho_2)}$$

$$\frac{yields(\tau_1,\sigma,\rho_1),\ yields(\tau_2,\sigma\bigoplus\rho_1,\rho_2)}{yields(\tau_1,\tau_2,\sigma,\rho_1\bullet\rho_2)}$$

mst semantics: $\nu(\tau) = \{\rho | \exists\theta : yield(\tau, \sigma, \theta, \rho)\}$

Semantics cont.

computation step

$\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ induces a transition $\sigma \rightarrow \sigma'$ iff $\sigma' = \sigma \bigoplus \rho$ and the program $\mathcal{P}$ yields ρ in σ, i.e. $\exists\theta : yields(\mathcal{P}, \sigma, \theta, \rho)$.

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.

BSM semantics (functional view)

$\nu(\tau) \rightsquigarrow$ a specification of *enabled* updates in states

▼

policies, code modularity

Semantics cont.

computation step

$\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ induces a transition $\sigma \rightarrow \sigma'$ iff $\sigma' = \sigma \bigoplus \rho$ and the program $\mathcal{P}$ yields ρ in σ, i.e. $\exists \theta : yields(\mathcal{P}, \sigma, \theta, \rho)$.

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.

BSM semantics (functional view)

$\nu(\tau) \rightsquigarrow$ a specification of *enabled* updates in states

▼

policies, code modularity

Semantics cont.

computation step

$\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ induces a transition $\sigma \rightarrow \sigma'$ iff $\sigma' = \sigma \bigoplus \rho$ and the program $\mathcal{P}$ yields ρ in σ, i.e. $\exists \theta : yields(\mathcal{P}, \sigma, \theta, \rho)$.

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.

BSM semantics (functional view)

$\nu(\tau) \rightsquigarrow$ a specification of *enabled* updates in states

↓

policies, code modularity

Abstract interpreter

input: agent program $\mathcal{P}$, initial mental state state σ_0

```
$\sigma = \sigma_0$
loop
    compute $\nu(\mathcal{P}) = \{\rho | \exists \theta : yields(\mathcal{P}, \sigma, \theta, \nu)\}$
    if $\nu(\mathcal{P}) \neq \emptyset$ then
        non-deterministically choose $\rho \in \nu(\mathcal{P})$
        $\sigma = \sigma \oplus \rho$
    end if
end loop
```

Example: office space robot

```
/* Initialization */
declare module beliefs as ASP
declare module goals as Prolog
declare module body as Java
/* initializations omitted */

/* Default operation */
when sense body [{ (Battery.status() == OK) }] then {
  /* Roam around */
  perform body [{ Motors.turn(Rnd.get(), Rnd.get()) }] ;
  perform body [{ Motors.stepForward() }]
} else
{
  /* Safe emergency mode — degrade gracefully */
  perform body [{ Face.smile(off) }] ;
  perform body [{ InfraEye.switch(off)}] ;
  update goals [{ assert(dock) }]
} ;
```

```
/* Interruption handling */
when sense body (X) [{ Visual.see(X) }] and
      query beliefs (X) [{ friend(X), not met(X) }]
then {
  perform body [{ Face.smile(on) }] ,
  perform body [{ Audio.say("Hello!") }] ,
  adopt beliefs (X) [{ met(X) }]
}
```

Jazzbot:

- **softbot in a simulated 3D environment** - *Nexuiz* game
- uses ASP module for beliefs and goals
- challenging, dynamic and rich environment

Example: office space robot

```
/∗ Initialization ∗/
declare module beliefs as ASP
declare module goals as Prolog
declare module body as Java
/∗ initializations omitted ∗/

/∗ Default operation ∗/
when sense body [{ (Battery.status() == OK) }] then {
  /∗ Roam around ∗/
  perform body [{ Motors.turn(Rnd.get(), Rnd.get()) }] ;
  perform body [{ Motors.stepForward() }]
} else
{
  /∗ Safe emergency mode — degrade gracefully ∗/
  perform body [{ Face.smile(off) }] ;
  perform body [{ InfraEye.switch(off)}] ;
  update goals [{ assert(dock) }]
} ;
```

```
/∗ Interruption handling ∗/
when sense body (X) [{ Visual.see(X) }] and
     query beliefs (X) [{ friend(X), not met(X) }]
then {
  perform body [{ Face.smile(on) }] ,
  perform body [{ Audio.say("Hello!") }] ,
  adopt beliefs (X) [{ met(X) }]
}
```

Jazzbot:

- **softbot in a simulated 3D environment** - *Nexuiz* game
- uses ASP module for beliefs and goals
- challenging, dynamic and rich environment

Summary

Jazzyk

An implemented programming language for agents with heterogeneous KRs:

- modularity ⇝ KR and source code - macros
- clear semantics ⇝ functional view(!)

- scope: single agent, non-critical applicationss:
  - videogames & virtual spaces
  - entertainment robotics
- prototyping in Jazzyk ⇝ test-bed for KRs in agents

Thank you for your attention.

http://jazzyk.sourceforge.net/

... thanks to Koen Hindriks ☺