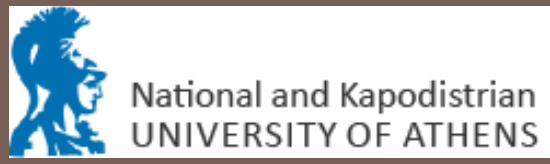
ACTION-BASED IMPERATIVE PROGRAMMING WITH YAGI



Alexander Ferrein, Gerald Steinbauer, Stavros Vassos

Yet Another Golog Interpreter

- Golog [Levesque; Reiter; Lesperance; Lin; Scherl, '97]
  - Agent programming language

Yet Another Golog Interpreter

- Golog [Levesque; Reiter; Lesperance; Lin; Scherl, '97]
  - Agent programming language
    - Can be used to specify the high-level behavior of robots, software bots, and systems using conditionals, loops, and more sophisticated features
    - Instead of operations on variables, it is based on actions that affect changing properties of the application domain

Domain description

High-level Program

Yet Another Golog Interpreter

- YAGI: aiming toward a (more) practical Golog implementation
  - Adopt a **familiar syntax** from popular software developing languages (C++, Java, Python, …)
  - Identify a **minimal subset** of the full Golog functionality that can be used by **non-experts**
  - Allow to be **easily embedded** to larger software projects and frameworks

The “smart elevator” example

- Simple running example to illustrate some of the basic functionality of Golog and YAGI

- Think of an **elevator** for a busy office center as an **agent** that follows a given **high-level plan**

  - Whenever there are pending floor requests pick one and serve the corresponding floor

Agent programming with Golog

- Logical domain description

- High-level program

Domain description

High-level Program

Agent programming with Golog

- Logical domain description

- High-level program
  - Specifies the behavior of the agent as a sketch of a plan using programming constructs

Domain description

High-level Program

Pick a floor with the button pressed and serve

Agent programming with Golog

- Logical domain description
  - Fluents: the changing properties of the domain
  - Actions: specify how the fluents change
- High-level program
  - Specifies the behavior of the agent as a sketch of a plan using programming constructs

Domain description

Fluents: CurrentFloor, On
Actions: Up, Down, TurnOff

High-level Program

Pick a floor with the button pressed and serve

Agent programming with Golog

- **Situation calculus**
  - **FOL language** specifically designed for reasoning about action and change
  - **Fluents:** Similar to database relations but also **conditioned on** a **situation** argument: F(x,y,$\mathbf{S_0}$)
  - $\mathbf{S_0}$ is the initial situation
  - **do(A,$\mathbf{S_0}$)** is the resulting situation after action A has been performed in S_0
  - Situations are used to refer to future states of the world F(x,y, **do(A,$\mathbf{S_0}$)**)

Agent programming with Golog

- Logical domain description
  - Fluent CurrentFloor(x,s): elevator is located at floor x
    - CurrentFloor(4,S_0)
  - Fluent On(x,s): button pressed at floor x
    - On(3,S_0), On(5,S_0)

Domain description

Fluents: CurrentFloor, On
Actions: Up, Down, TurnOff

Agent programming with Golog

- Logical domain description
  - Action Up(x)/Down(x): move up/downwards to floor x
  - CurrentFloor(x,**do(a,s)**) $\equiv$ a=Up(x) $\vee$ a=Down(x)
    $\vee$ CurrentFloor(x,**s**) $\wedge \neg\exists$y a=Up(y) $\wedge \neg\exists$y a=Down(y)
  - Poss(Up(x),**s**) $\equiv \exists$x (CurrentFloor(y,**s**) $\wedge$ y<x)

Domain description

Fluents: CurrentFloor, On
Actions: Up, Down, TurnOff

Agent programming with Golog

- High-level program
  - Specifies the behavior of the agent as a sketch of a plan using programming constructs
- π x. ((Up(x) | Down(x));
  TurnOff(x);
  open();
  close())

High-level Program

Pick a floor with the button pressed and serve

Agent programming with Golog

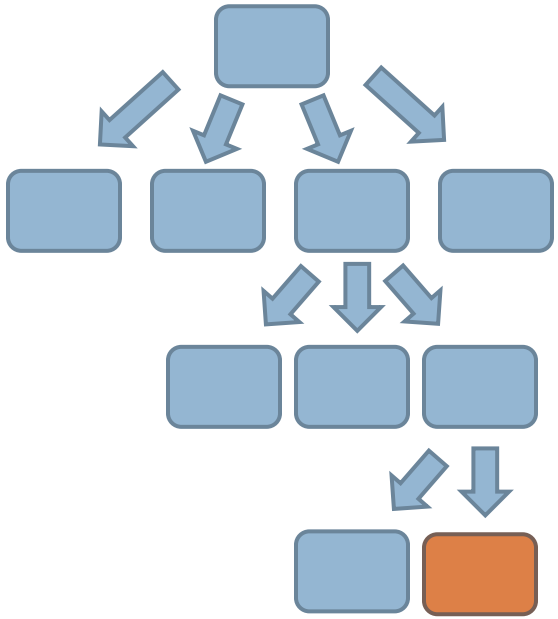
- High-level program
  - Specifies the behavior of the agent as a sketch of a plan using programming constructs
- π x. ((Up(x) | Down(x));
  TurnOff(x);
  open();
  close())
- ServeAFloor()

High-level Program

Pick a floor with the button pressed and serve

Agent programming with Golog

- High-level program
  - **Off-line** vs **On-line** execution



- π x. ((Up(x) | Down(x));
  TurnOff(x);
  open();
  close())

Domain description

Fluents: CurrentFloor, On
Actions: Up, Down, TurnOff

High-level Program

Pick a floor with the button pressed and serve

Agent programming with Golog

- Golog programming constructs

α,	primitive action
$\phi?$,	wait or test for a condition
$\delta_1; \delta_2$,	sequence
$\delta_1 \mid \delta_2$,	nondeterministic branch
$\pi\, x.\, \delta(x)$,	nondeterministic choice of argument
δ^*,	nondeterministic iteration
if ϕ **then** δ_1 **else** δ_2 **endIf**,	conditional
while ϕ **do** δ **endWhile**,	while loop
$\delta_1 \parallel \delta_2$,	concurrency with equal priority
$\delta_1 \rangle\!\rangle \delta_2$,	concurrency with δ_1 at a higher priority
$\delta^{\parallel}$,	concurrent iteration
$\langle\, \vec{x} : \phi(\vec{x}) \longrightarrow \delta(\vec{x}) \,\rangle$,	interrupt
$p(\vec{\theta})$.	procedure call

Agent programming with Golog

- Logical domain description
  - Precise logical formulation based on situation calculus
- High-level program
  - Precise logical formulation based on Golog
- Typically **implemented** on top of some **Prolog** interpreter

Domain description

Fluents: CurrentFloor, On
Actions: Up, Down, TurnOff

High-level Program

Pick a floor with the button pressed and serve

Agent programming with Golog

- Programming requires understanding the semantics
  - CurrentFloor(x,**do(a,s)**) ≡ a=Up(x) ∨ a=Down(x) ∨ CurrentFloor(x,**s**) ∧ ¬∃y a=Up(y) ∧ ¬∃y a=Down(y)
  - Poss(Up(x),**s**) ≡ ∃x (CurrentFloor(y,**s**) ∧ y<x)
  - π x. ((Up(x) | Down(x)); TurnOff(x); open(); close())
- Features of Prolog are implicitly used in the domain specification and the high-level program

Agent programming with Golog

- Programming requires understanding the semantics
  - CurrentFloor(x,**do(a,s)**) ≡ a=Up(x) ∨ a=Down(x) ∨ CurrentFloor(x,**s**) ∧ ¬∃y a=Up(y) ∧ ¬∃y a=Down(y)
  - Poss(Up(x),**s**) ≡ ∃x (CurrentFloor(y,**s**) ∧ y<x)
  - π x. ((Up(x) | Down(x)); TurnOff(x); open(); close())
- Features of Prolog are implicitly used in the domain specification and the high-level program
- **Confusing** in practice for **non-expert users**

Yet Another Golog Interpreter

- Aiming toward a (more) practical Golog implementation
  - First iteration (this paper): **YAGI language specification** as a stand-alone interpreted programming language with familiar syntax

Yet Another Golog Interpreter

- YAGI language
  - Make **fluents** look like **variables** of popular programming languages
  - Make **actions** look like **functions** of popular programming languages
- Unify the domain description and high-level program using this metaphor

Yet Another Golog Interpreter

- Make **fluents** look like **variables** of popular programming languages
  - CurrentFloor(2,S_0)
  - ```
    YAGI>> fluent CurrentFloor;
    YAGI>> On = {2};
    ```
  - Dictionary/Associative array/Map
    - Associates to a flat list of values
    - Multi-dimensional key (here On is 0-dimensional)
    ```

Yet Another Golog Interpreter

- Make **fluents** look like **variables** of popular programming languages
  - On(3,S_0), On(5,S_0)
  - **YAGI>> fluent On;**
    YAGI>> On = {1,3}
  - $\forall x$ (On(x,S_0) $\equiv$ (x=1 $\vee$ x=3))
  - Note that we (currently) aim for complete knowledge

Yet Another Golog Interpreter

- Make **fluents** look like **variables** of popular programming languages
  - Set-theoretic operations on the list of values
  - ```
    YAGI>> On += 5
    YAGI>> On
    {1,3,5}
    YAGI>> On -= 1
    {3,5}
    YAGI>> 3 in On
    True
    ```
    ```

Yet Another Golog Interpreter

- Make **actions** look like **functions** of popular programming languages
  - Action Up(x)
  - ```
    YAGI>> action Up($n)
      ...
    end action
    ```
  - Specify local variables using a $ prefix
  - Specify preconditions and effects as YAGI statements between fluents
    ```

Yet Another Golog Interpreter

- Make **actions** look like **functions** of popular programming languages
  - Action Up(x)
  - ```
    YAGI>> action Up($n)
    precondition:
      ...
    effect:
      currentFloor = {$n}
    end action
    ```
    ```

Yet Another Golog Interpreter

- Make **actions** look like **functions** of popular programming languages
  - Action Up(x)
  - ```
    YAGI>> action Up($n)
    precondition:
      ...
    effect:
      for $i in On do
            currentFloor += $i
      end for
    end action
    ```
    ```

Yet Another Golog Interpreter

- Use the **same syntax**/metaphor to specify the **high-level program**

```
YAGI>> proc serveAFloor()
  pick $n from On such
    choose
       up($n)
    or
       down($n)
    end choose
    turnOff($n); open(); close();
  end pick
end proc
```

Yet Another Golog Interpreter

- High-level program calls: Each YAGI call is **searched offline** and then **executed online**

```
YAGI>> On
{3,5}
YAGI>> serveAFloor()
YAGI>> On
{5}

YAGI>> serveAFloor()
YAGI>> On
{}
```

Yet Another Golog Interpreter

- High-level program calls: Each YAGI call is **searched offline** and then **executed online**

```
YAGI>> On
{3,5}
YAGI>> serveAFloor(); serveAFloor()
YAGI>> On
{}
```

Yet Another Golog Interpreter

- **Interoperability** using **string signals**
- (Inspired from web services, aiming for XML-like or JSON-like interoperability)

```
YAGI>> action Up($n)
precondition:
  ...
effect:
  ...
signal:
 "Move up to floor " + $n
end action
```

Yet Another Golog Interpreter

- High-level program calls: Each YAGI call is **searched offline** and then **executed online**

```
YAGI>> serveAFloor()
"Move up to floor 5"
"Turn-off button at floor 5"
"Open elevator door"
"Close elevator door"

YAGI>> serveAFloor()
"Move up to floor 3"
"Turn-off button at floor 3"
"Open elevator door"
"Close elevator door"
```

Yet Another Golog Interpreter

- High-level program calls: Each YAGI call is **searched offline** and then **executed online**

```
YAGI>> serveAFloor(); serveAFloor()
"Move up to floor 5"
"Turn-off button at floor 5"
"Open elevator door"
"Close elevator door"
"Move up to floor 3"
"Turn-off button at floor 3"
"Open elevator door"
"Close elevator door"
```

Situation calculus semantics

- YAGI operates as a persistent object that
  - Holds a situation calculus basic action theory
  - Responds to a YAGI call by
    - finding an appropriate sequence of actions to perform following Golog semantics
    - updating the theory by progressing the initial knowledge based executing each action one by one
    - producing signals along the way
  - Responds to a YAGI declaration by updating directly the specification of the theory

Situation calculus semantics

- Differences with the family of Golog languages
  - Complete knowledge
  - No while loops in the language (only for loops)
  - Addition and equality support with Presburger arithmetic
  - Progression is built-in the semantics

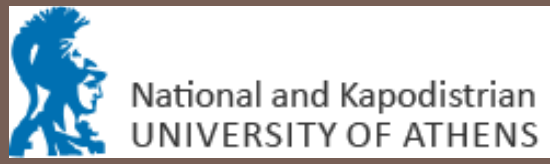
Current and future work

- Aiming toward a (more) practical Golog implementation
  - First iteration (this paper): **YAGI language specification** as a stand-alone interpreted programming language with familiar syntax
    - YAGI BNF spefication in the paper
    - Parser available online: http://code.google.com/p/yagi/
  - Second iteration (current and future work): **YAGI prototype interpreter**
    - C++, Java, Lua
    - On top of existing Prolog implementation

Conclusions

- YAGI: First formal specification of Golog as a non logic-based programming language
- A roadmap for some essential requirements
  - Non-functional requirements
    - Q1: Familiarity
    - Q2: Embeddedness
    - Q3: Interoperability
    - Q4: Transparency
  - Functional requirements
    - F1-F12: Action-based, imperative, goal-oriented, arithmetic, queries, null values, probabilistic values, sensing, …

ACTION-BASED IMPERATIVE PROGRAMMING WITH YAGI



Alexander Ferrein, Gerald Steinbauer, Stavros Vassos