ActHex

HEX Programs with Action Atoms

STEFANO GERMANO

Brief Summary

- HEX Programs
  - Syntax
  - Semantics
  - Example
  - Implementation
- ACTHEX Programs
  - Syntax
  - Semantics
  - Example
  - Implementation
  - Demo (SmartRobot)
- Belief–Desire–Intention software model in ACTHEX Programs

Syntax of HEX Programs

- $\mathcal{C}$ are constants;
- $\mathcal{X}$ are variable names;
- $\mathcal{C} \cup \mathcal{X}$ are terms;
- $\mathcal{G}$ are external predicate names

- A (higher-order) atom is a tuple $Y_0(Y_1,\ldots,Y_n)$, all Y_i are terms.

An external atom is of the form:

$$\&g[Y_1,\ldots,Y_n](X_1,\ldots,X_m)$$

- $\&g \in \mathcal{G}$ is an external predicate name
- $Y_1,\ldots,Y_n$ is the input list, Y_i are terms
- $X_1,\ldots,X_m$ is the output list, X_i are terms

Syntax of HEX Programs

A rule $\boldsymbol{r}$ is of the form:

$$\alpha_1 \vee \ldots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_m, \mathit{not}\ \beta_{m+1}, \ldots, \mathit{not}\ \beta_n$$

- $\alpha_1, \ldots, \alpha_k$ are atoms
- $\beta_1, \ldots, \beta_n$ are atoms or External Atoms

A HEX program (or program) is a finite set $\boldsymbol{\mathcal{P}}$ of rules.

Semantics of HEX Programs

Interpretation of $\mathcal{P}$: subset $I \subseteq \mathcal{HB}_{\mathcal{P}}$ containing only ordinary atoms;

I is a model of

- an atom $\boldsymbol{\alpha} \in \mathcal{HB}_{\mathcal{P}}$ ($I \models \boldsymbol{\alpha}$), if $\boldsymbol{\alpha} \in I$.
- a ground external atom $\boldsymbol{\alpha}=\&g[c_2](c_1)$ ($I \models \alpha$), if $f_{\&g}(I,c_2,c_1)=1$
  - $f_{\&g}$ is a (fixed) $(n+m+1)$-ary Boolean function for $\&g$,
  - where $n=in(\&g)$, $m=out(\&g)$, and $c_2 \in \mathcal{C}^n$, $c_1 \in \mathcal{C}^m$.
- a ground rule r ($I \models r$), if $I \models \mathcal{H}(r)$ or $I \not\models \mathcal{B}(r)$.
- a program $\mathcal{P}$ ($I \models \mathcal{P}$), iff $I \models r$ for all $r \in grnd(\mathcal{P})$.

Example of HEX Programs

Consider an external atom *&reach[G,X](Y)* computing reachability in a directed graph.

$f_{\&reach}(\mathcal{I},\mathcal{G},X,Y) = 1$ iff *Y* is reachable from *X* in the directed graph encoded by the extension of binary predicate $\mathcal{G}$ in $\mathcal{I}$.

Example (Computing paths)

arc(1,2). arc(2,3).
node(X) ← arc(X,Y).
node(Y) ← arc(X,Y).
path(X,Y) ← &reach[arc,X](Y),node(X).

Its unique answer set includes {*path(1,2),path(2,3),path(1,3)*}.

Implementation of HEX Language

dlvhex is the name of a prototype application for computing the models of HEX-programs.

http://www.kr.tuwien.ac.at/research/systems/dlvhex/

https://github.com/hexhex/core

Various HEX plugins exist: stringplugin, wordnetplugin, mcsieplugin, dlplugin, scriptplugin, xpathplugin, mathematicaplugin etc.

Syntax of ACTHEX Programs

- Mutually disjoint sets $\mathcal{C}$, $\mathcal{X}$, $\mathcal{G}$ (as before);
- $\mathcal{A}$ are action predicate names;
- Ordinary, higher order, external atoms are defined as before.

An action atom is of the form:

$$\#g[Y_1,\ldots,Y_n]\{o,r\}[w{:}l]$$

- $\#g \in \mathcal{A}$ is an action predicate name
- $Y_1,\ldots,Y_n$ is the input list, Y_i are terms
- $o \in \{b,c,c_p\}$ is called the action option
- r denotes precedence
- w and l denote weight, and level

Syntax of ACTHEX Programs

A rule $\boldsymbol{r}$ is again of the form:

$$\alpha_1 \vee \ldots \vee \alpha_k \leftarrow \beta_1 , \ldots , \beta_m , \mathit{not}\ \beta_{m+1} , \ldots , \mathit{not}\ \beta_n$$

- $\alpha_1 , \ldots , \alpha_k$ are atoms or Action Atoms
- $\beta_1 , \ldots , \beta_n$ are atoms or External Atoms

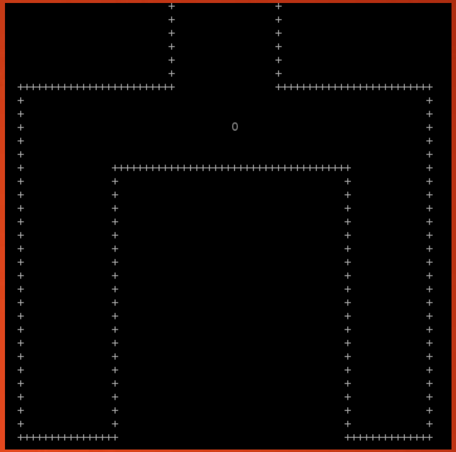
An ACTHEX program (or program) is a finite set $\mathcal{P}$ of rules.

Semantics of ACTHEX Programs

The Semantics derives from the Semantics of HEX Programs with some extensions.

Intuitively defined with an example in the next slide.

Example of ACTHEX Programs



Consider a Robot standing at the center of a U shaped corridor.

Example (Robot)

fuel(X) :- &sense[fuel](X).
weights(Wl, Wr) :- &weights[](Wl, Wr).
#robot[move,all]{b,1} :- fuel(high).
#robot[move,left]{b,1}[Wl :] v #robot[move,right]{b,1}[Wr :] :- fuel(low), weights(Wl, Wr).

Its answer set(s) depend(s) on the value of the External Atoms

Implementation of ACTHEX Language

An implementation of ACTHEX programs has been realized as an extension of *dlvhex* called *ActionPlugin*

http://www.kr.tuwien.ac.at/research/systems/dlvhex/actionplugin.html

https://github.com/hexhex/actionplugin

Exist some example Addons of *ActionPlugin*: Robot, KBMod, BoolMatrix, SmartRobot, Sudoku.

High-Level Architecture

Module that can be customized by the users

- Addons
- Environment
- BestModel Selector
- Execution Schedule Builder
- Iterator

ActionPlugin Addons

An interface (ActionPluginInterface) makes possible the creation of Addons for Action Plugin.

The creation of Addons is extremely simple and fast.

Each Addon has (and can customize in a very simple way) an Environment, it is shared between the Action Atoms and the External Atoms of the same Addon, it also retains its state during various iterations.

In addition, each Addon can include an arbitrary number of Action Atoms, External Atoms, BestModel Selectors, Execution Schedule Builders and through some functions the creator of the Plugin can specify which of these use.

Environment

The *Environment* is a class that every Addon can use (each has its own *Environment*) as a knowledge base. The *Environment* can be read by External Atoms and can be read and edited by Action Atoms.

Have this space, jointly with the possibilities offered by the *Iteration* is very important for an *Addon* because allows it to store information, process this information and then change its behavior in subsequent iterations.

BestModel Selector

The user can specify a custom *BestModel Selector* using a Built-in Constant.

The Built-in Constant is *#acthexBestModelSelector* must have as a parameter the name of the *BestModel Selector* that the user want to use.

For example:

#acthexBestModelSelector = bestModelSelector1.

Execution Schedule Builder

The user can specify a custom *Execution Schedule Builder* using a Built-in Constant.

The Built-in Constant is *#acthexExecutionScheduleBuilder* must have as a parameter the name of the *Execution Schedule Builder* that the user want to use.

For example:

#acthexExecutionScheduleBuilder = executionScheduleBuilder1.

Iterator

There is the possibility to iterate the process of evaluation/execution of Action Atoms.

An Iteration is the evaluation of a given HEX program with Action Atoms and the execution of the entailed Action Atoms.

This feature is very useful and becomes necessary when we want to use *Environment* in a proper way.

Iterator

The behavior of the iteration process is controlled by means of a Iteration variable.

The default value of Iteration is “NO ITERATION” meaning that we won’t do any iteration namely the evaluation/execution of Action Atoms will be performed only one time.

The user can modify this default behavior in 3 different ways:

- from a command line option
- by changing some built-in constant values (that override the behavior specified from command line)
- by entailing the truth of some built-in action predicates.

Existing ActionPlugin Addons

- Robot
- KBMod
- BoolMatrix
- SmartRobot
- Sudoku

Potential Applications of ACTHEX Programs

- Knowledge Base Updates
- Action Languages
- Logic-based Agent Programs

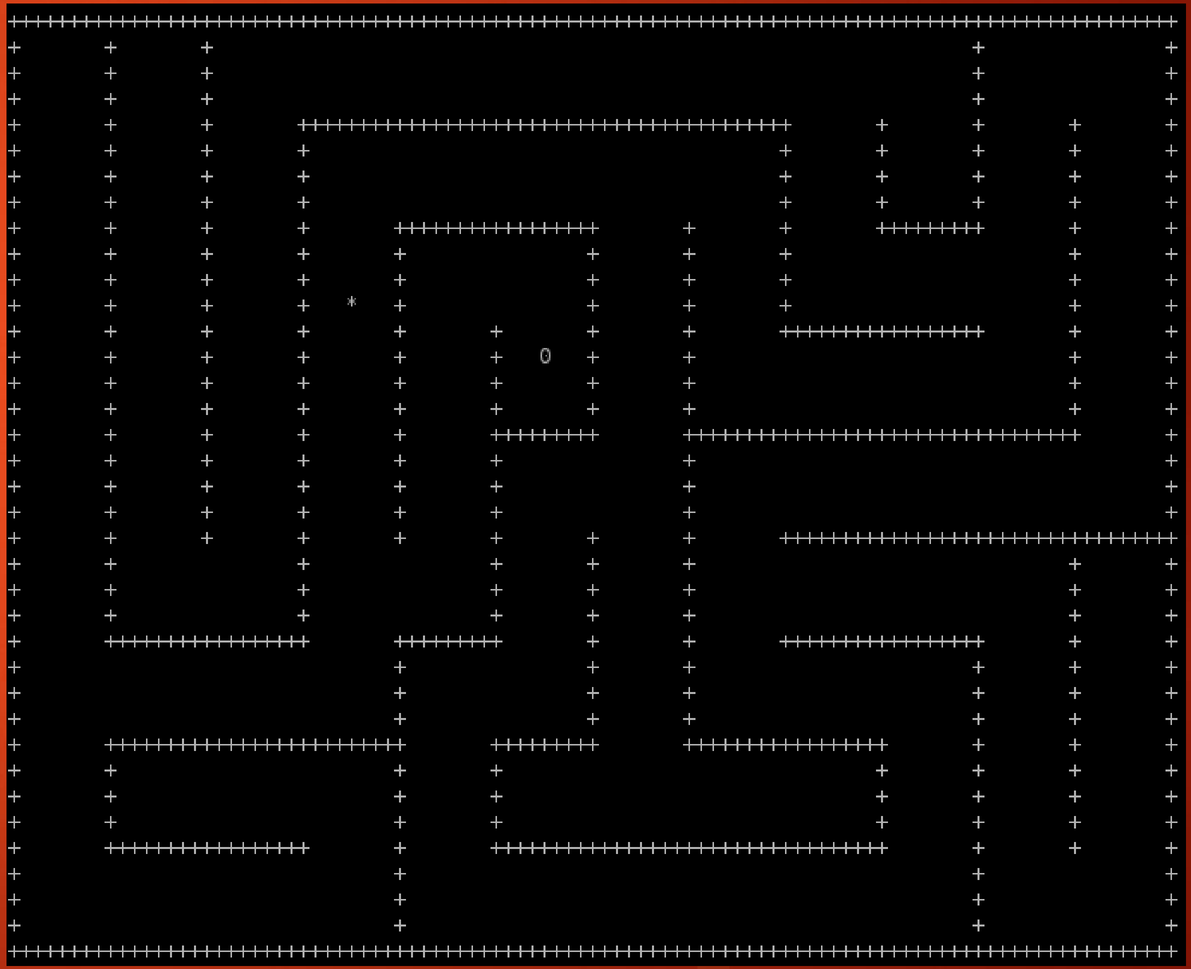
Demo (SmartRobot)

Overview

Consider a Robot in a Maze that has to find a treasure inside the Maze.

It can move only in one of the 4 cells around it and can recognize if there is a wall.

It tries to do this ins a smart way, trying to not go back into the cells where it has already been.



Demo (SmartRobot)

Example of External Atom

```
class SmartRobotActionAtomGetPosition : public PluginActionAtom<SmartRobotActionPlugin> {
public:
        SmartRobotActionAtomGetPosition() : PluginActionAtom("getPosition") { setOutputArity(2); }
private:
        void retrieve(const Environment& environment, const Query& query, Answer& answer) {
                Registry &registry = *getRegistry();
                if (query.input.size() != 0) throw PluginError("Wrong input argument type");
                std::stringstream concatstreamRow; concatstreamRow << environment.getCurrentPositionRow();
                std::stringstream concatstreamColumn; concatstreamColumn << environment.getCurrentPositionColumn();
                Tuple out;
                Term termRow(ID::MAINKIND_TERM | ID::SUBKIND_TERM_CONSTANT, std::string(concatstreamRow.str()));
                Term termColumn(ID::MAINKIND_TERM | ID::SUBKIND_TERM_CONSTANT, std::string(concatstreamColumn.str()));
                out.push_back(registry.storeTerm(termRow)); out.push_back(registry.storeTerm(termColumn));
                answer.get().push_back(out);
        }
};
```

Demo (SmartRobot)

Example of Action Atom

```
class SmartRobotAction: public PluginAction<SmartRobotActionPlugin> {
public:
	SmartRobotAction() : PluginAction("smartRobot") {}
private:
	void execute(Environment& environment, RegistryPtr pregistry, const Tuple& parms, const InterpretationConstPtr interpretationPtr) {
		Registry& registry = *pregistry;
		if (registry.getTermStringByID(parms[0]) == "move") {
			if (registry.getTermStringByID(parms[1]) == "up") environment.moveUp();
			else if (registry.getTermStringByID(parms[1]) == "down") environment.moveDown();
			else if (registry.getTermStringByID(parms[1]) == "left") environment.moveLeft();
			else if (registry.getTermStringByID(parms[1]) == "right") environment.moveRight();
			else throw PluginError("Unknown move!");
			environment.refresh(); }
	}
};
```

Demo (SmartRobot)

Example of ActHex code

```
#smartRobot[move,up,0]{b,10}[Wup:] v
       #smartRobot[move,down,0]{b,10}[Wdown:] v
       #smartRobot[move,left,0]{b,10}[Wleft:] v
       #smartRobot[move,right,0]{b,10}[Wright:] :- canMoveUp, canMoveDown, canMoveLeft, canMoveRight,
                                                   &getNumberOfTimes[](Wup,Wdown,Wleft,Wright).

canMoveUp :- current_position(R,C), R != 0, not wall(R_m_1,C), R = R_m_1 + 1, row(R_m_1).
canMoveDown :- current_position(R,C), Rs = Rs_m_1 + 1, R != Rs_m_1, rows(Rs), row(Rs_m_1),
               not wall(R_p_1,C), R_p_1 = R + 1, row(R_p_1).
canMoveLeft :- current_position(R,C), C != 0, not wall(R,C_m_1), C = C_m_1 + 1, column(C_m_1).
canMoveRight :- current_position(R,C), Cs = Cs_m_1 + 1, C != Cs_m_1, columns(Cs), column(Cs_m_1),
                not wall(R,C_p_1), C_p_1 = C + 1, column(C_p_1).

current_position(R,C) :- &getPosition[](R, C).
```

Live Demo

Belief–Desire–Intention software model in ActHex Programs

Beliefs

- Facts
- External Atoms

Desires

- Weights and Levels in Action Atoms

Intentions

- Action Atoms

Questions?

Special thanks go to Prof. Giovambattista Ianni and to the Knowledge-Based Systems Group of the Vienna University of Technology, in particular Dr. Michael Fink, Dr. Thomas Krennwallner, Dr. Christoph Redl and Dr. Peter Schüller.

Thank you for your attention