ActHEX: Implementing HEX Programs with Action Atoms[⋆]

Michael Fink[1], Stefano Germano[2], Giovambattista Ianni[2],
Christoph Redl[1], and Peter Schüller[3]

[1] Institut für Informationssysteme, Technische Universität Wien
[2] Dipartimento di Matematica e Informatica, Università della Calabria
[3] Faculty of Engineering and Natural Sciences, Sabanci University

Abstract. acthex programs are a convenient tool for connecting stateful external environments to logic programs. In the acthex framework, actual actions on an external environment can be declaratively selected, rearranged, scheduled and then executed depending on intelligence specified in an ASP-based language. We report in this paper about recent improvements of the formal and of the operational acthex programming framework. Besides yielding a significant increase in versatility of the framework, we also present illustrative application showcases and a short evaluation thereof exhibiting computational acthex strengths.

1 Introduction

The acthex formalism [1] generalizes HEX programs [4] introducing dedicated action atoms in rule heads. Action atoms can actually operate on and change the state of an *environment*, which can be roughly seen as an abstraction of realms outside the logic program at hand. The acthex framework allows to conveniently design ASP-based applications by properly connecting logic-based decisions to actual effects thereof. We recently advanced the acthex framework wrt. several respects:

– Framework improvements: external atom evaluation has been generalized to take state into account, i.e., the realm of acthex programs has been extended to capture nondeterministic actions and environments. Moreover, support for selecting a single model and a unique corresponding execution schedule has been enhanced, and we developed explicit means for controlling iterative evaluation of logic programs.
– System improvements: we provide a new architecture for the acthex framework efficiently implemented as an extension to the dlvhex system[4].
– Applications: we realized new applications and pursued a preliminary system evaluation exhibiting promising results. In terms of performance, our experiments indicate that, compared to purely declarative approaches, finding problem solutions iteratively may pay off when instances are large. In terms of ease of programming, our approach allows to attach code in arbitrary programming languages to a logic-programming framework. This is dual to other approaches to interoperability of ASP solvers like [5].

[⋆] This research has been partially supported by the Vienna Science and Technology Fund (WWTF) grant ICT 08-020 Austrian Science Fund (FWF) grant P24090 Peter Schüller is supported by the TUBITAK 2216 Research Fellowship.

[4] Available at http://www.kr.tuwien.ac.at/research/systems/dlvhex/actionplugin.html

2 Preliminaries

We assume familiarity with ASP and corresponding basic syntactic and semantic notions (atoms, models, etc.). For space reasons, in the following, we also do not present acthex syntax and semantics at full formal detail (for the latter cf. [1,4]).

acthex syntax. In addition to constants (also used for predicate names) and variables, acthex programs build on external predicate names (prefixed by &) and action predicate names (prefixed by #). An external atom is of the form $\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m)$, where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are lists of terms. An action atom is of the form $\#g[Y_1, \ldots, Y_n]\{o, r\}[w : l]$, where $\#g$ is an action predicate name, $Y_1, \ldots, Y_n$ is a list of input terms of fixed length $in(\#g) = n$. Moreover, attribute $o \in \{b, c, c_p\}$ is called the *action option* that identifies an action as *brave, cautious, or preferred cautious*, while optional integer attributes r, w, and l are called *precedence*, *weight*, and *level* of $\#g$, respectively. A *rule* r is of the form $\alpha_1 \vee \ldots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, \text{not } \beta_{n+1}, \ldots, \text{not } \beta_m$, where body elements β are (ordinary) atoms or external atoms, and head elements α are (ordinary) atoms or action atoms. An acthex *program* is a finite set of rules.

Example 1. The acthex program $P_1 = \{\#robot[goto, charger]\{b, 1\} \leftarrow \&sensor[bat](low);$ $\#robot[clean, kitchen]\{c, 2\} \leftarrow night; \#robot[clean, bedroom]\{c, 2\} \leftarrow day; night \vee day \leftarrow\}$ uses action atom $\#robot$ to control a robot, and an external atom $\&sensor$ to access sensor data. Intuitively, precedence 1 of action atom $\#robot[goto, charger]\{b, 1\}$ should make the robot recharging its battery, if necessary, before cleaning actions. $\square$

acthex semantics. An acthex program P is evaluated wrt. a fixed state (snapshot) of the *external environment* E using the following steps: (i) *answer sets* of P are determined wrt. E, and the set of *best models* is a subset of the answer sets determined by an objective function; (ii) any (best) model originates a set of corresponding *execution schedules* S, i.e., a sequence of actions to execute; (iii) executing the actions of (and sequentially according to) a selected schedule S yields another (not necessarily different) state E' of the environment, called the *observed execution outcome*; finally (iv) the process may be iterated starting at (i), by considering a snapshot E'', which can be different from E' due to exogenous actions (in so-called dynamic environments). Answer Sets are defined similarly to HEX programs [4], i.e., using Herbrand interpretations, the grounding of P wrt. the Herbrand universe, and the FLP reduct; ground action atoms in rule heads are treated like ordinary atoms, see Section 3 for a generalized external atom semantics including the environment E. We denote by $\mathcal{AS}(P, E)$ the collection of all answer sets of P wrt. E. The set of *best models* of P, denoted $\mathcal{BM}(P, E)$, contains those answer sets $I \in \mathcal{AS}(P, E)$ that minimize an objective function over weights and levels of atoms in I (equivalent to the evaluation of weak constraints in [2]). An action $a = \#g[y_1, \ldots, y_n]\{o, r\}[w : l]$ with option o and precedence r is *executable in* I *wrt.* P *and* E iff (i) a is brave and $a \in I$, or (ii) a is cautious and $a \in B$ for every $B \in \mathcal{AS}(P, E)$, or (iii) a is preferred cautious and $a \in B$ for every $B \in \mathcal{BM}(P, E)$. An *execution schedule* S_I for a (best) model I is a sequence of all actions executable in I, such that for all pairs of action atoms $a, b \in I$, if $prec(a) < prec(b)$ then a must precede b in S_I, for $prec(c)$ the precedence of an action atom c. Concerning the effects of actually executing actions, as well as corresponding notions of execution outcomes, we also refer to the next section where these notions are generalized compared to definitions in [1].

Example 2. Considering the program of Example 1, if the robot has low battery, then $\mathcal{AS}(P, E) = \mathcal{BM}(P, E)$ contains two models:

$$I_1 = \{night, \#robot[clean, kitchen]\{c, 2\}, \#robot[goto, charger]\{b, 1\}\}, \text{ and}$$
$$I_2 = \{day, \#robot[clean, bedroom]\{c, 2\}, \#robot[goto, charger]\{b, 1\}\}.$$

Both give rise to a single execution schedule S_{I_i}: first charge, then clean. □

3 Conceptual Improvements to the acthex framework

The effective implementation of acthex within the dlvhex software, as well as its initial application and preliminary evaluation (cf. Sections 4 and 5), raised practical issues calling for conceptual changes of the acthex framework. Compared to its definition in [1], we incorporated the following improvements.

External Atom and Action Atom Semantics. We *generalize external atom semantics* in order to take the environment into account as follows. With every external predicate name *&g* we associate an $(n+m+2)$-ary Boolean function $f_{\&g}$, assigning each tuple $(E, I, y_1, \ldots, y_n, x_1, \ldots, x_m)$ either 0 or 1, where E is an environment state, I an interpretation, and the other parameters are input and output constants of *&g*, respectively. We say that an interpretation I relative to P is a *model* of a ground external atom $a = \&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$ wrt. environment E, denoted as $I, E \models a$, iff $f_{\&g}(E, I, y_1 \ldots, y_n, x_1, \ldots, x_m) = 1$.

Given a model I, for each action predicate name *#g* the *possible effects of executing a ground action* $\#g[y_1, \ldots, y_m]\{o, p\}[w : l]$ on an environment E wrt. I are defined by an associated $(m+2)$-ary function $f_{\#g}$ which returns a set of possible follow-up environment states: $f_{\#g}(E, I, y_1, \ldots, y_m) = \mathcal{E}$. Every $E' \in \mathcal{E}$ thus represents a possible effect. Considering a set of environments rather than a definite effect allows to model nondeterministic actions, and also nondeterministic and/or dynamic environments, where the environment may change without action execution by means of exogenous events.

Model Selection and Execution Schedule Representation. In practice, one usually wants to consider and execute a single execution schedule. This requires the choice of a single best model and a unique corresponding execution schedule. The former is modelled by a *Best Model Selector* function $select_{BM}$ which intuitively decides which model I from $\mathcal{BM}(P, E)$ to use. In our implementation, some simple selection functions (like lexicographic first) are built-in and can be configured. Alternatively, $select_{BM}$ can be provided in terms of user-defined C++ code. The set of all execution schedules of I is given by $\mathcal{ES}_{P,E}(I) =$

$$\big\{\langle a_1, \ldots, a_n\rangle \mid prec(a_i) \le prec(a_j), \text{ for } 1 \le i < j \le n, \text{ and } \{a_1, \ldots, a_n\} = A_e\big\}.$$

$\mathcal{ES}_{P,E}(I)$ is in principle as large as $\mathcal{O}(|I|!)$, thus it is of course represented implicitly by its *execution schedule base* $\mathcal{ESB}_{P,E}(I)$, which is defined as a sequence of sets of actions $\mathcal{ESB}_{P,E}(I) = \big\{\langle A_1, \ldots, A_m\rangle\big\}$ where $A_i \subseteq A_e$, $1 \le i \le m$, and $prec(a) = prec(a')$ for all $a, a' \in A_i$, while $prec(a) < prec(a'')$ holds for all $a \in A_i, a'' \in A_j, 1 \le j \le m, i \ne j$. Intuitively, actions in A_i have the same precedence, while the precedence of actions strictly increases along the sequence. Obviously, $\mathcal{ESB}_{P,E}(I)$ has size $\mathcal{O}(|I|)$, and $\mathcal{ES}_{P,E}(I)$ can be recovered from it.

Execution Schedule Selection and Execution Outcomes. Given an execution schedule base, again a particular (customizable) function build_{ES}, called *Execution Schedule Builder*, selects a single execution schedule $\langle a_1, \ldots, a_n\rangle \in \mathcal{ES}_{P,E}(I)$ for execution. It defines a strict order over potential schedules, possibly based on general criteria on actions independent from the current execution base. Given an execution schedule $S = \langle a_1, \ldots, a_n\rangle \in \mathcal{ES}_{P,E}(I)$, the set of *possible execution outcomes* of S in environment E wrt. I is defined as $\mathit{EX}(S, I, E) = \{E_n \mid E_0 = E, \text{ and } E_{i+1} \in f_{\#g}(E_i, I, y_1, \ldots, y_m)\}$, given that a_i is of the form $\#g[y_1, \ldots, y_m]\{o, p\}[w : l]$. Intuitively the initial environment $E_0 = E$ is succeeded by a potential effect of executing each action in S in the given order. Recall that nondeterministic functions $f_{\&g}$ not only capture nondeterministic actions but take into account nondeterministic and/or dynamic environments. Eventually, given acthex program P, environment E, execution schedule builder build_{ES} and best model selector select_{BM}, the *observed outcome* of executing P on E is given by some $E_n \in \mathit{EX}(S, I, E)$, where $S = \mathit{build}_{ES}(\mathcal{ESB}_{P,E}(I))$ and $I = \mathit{select}_{BM}(\mathcal{BM}(P, E))$. Unless the environment is static and deterministic, from a modeling perspective, the observed outcome represents a nondeterministic choice. For instance, executing S_{I_1} of Example 2 assuming a static and deterministic environment first yields $\{E_1\} = f_{\#robot}(E, I_1, \mathit{goto}, \mathit{charger})$, and then $\{E_2\} = f_{\#robot}(E_1, I_1, \mathit{clean}, \mathit{bedroom})$, where E_2 is the observed execution outcome.

Evaluation Iteration. Another important implementation aspect is an efficient realization of iterative acthex program evaluation. For this purpose we provide support on two aspects. First, in order to capture systems with dynamic environments, the environment state is sensed upon each iteration. This yields the environment $E = E_0$ that is used to evaluate external atoms and upon which the first scheduled action is executed. In general, the environment $E = E'_i$ for evaluation in iteration $i + 1$ can possibly differ from the observed outcome E_i at iteration i. Second, iteration control is provided by dedicated command line options, built-in constants, and specific action atoms. From the command line, and with higher priority by setting built-in constants to true, one can effect iterative evaluation in terms of fixed number of iterations, iteration until a pre-set value of (total) execution time is elapsed (checked after each iteration), and iteration ad infinitum. Special action atoms $\#\mathit{acthexContinue}$ and $\#\mathit{acthexStop}$ have highest priority and provide a declarative means of controlling iteration.

4 System Architecture and Implementation

Figure 1 shows how acthex is implemented within the dlvhex [4] framework, how applications interface with acthex, and the stages of executing an acthex program.

A given acthex program P is first parsed using the dlvhex parser, the acthex-specific parser, and the program rewriter modules. This yields a HEX program P' which contains auxiliary atoms instead of actions. P and P' are such that the sets $\mathcal{AS}(P')$ and $\mathcal{AS}(P, E)$ are in one-to-one correspondence. P' is evaluated using the computational core of dlvhex wrt. *custom* external atoms of an acthex application, then the set of best models $\mathcal{BM}(P, E)$ is computed. One best model I is selected using a Best Model Selector

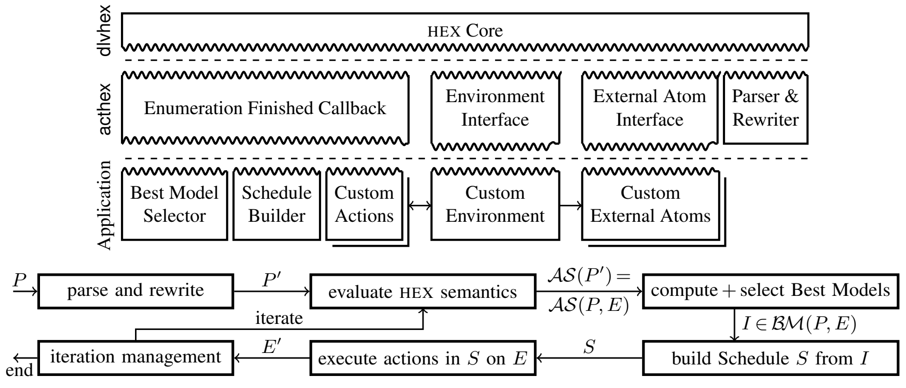

Fig. 1: Architecture of acthex and execution flow for program P on Environment E.

module, then an Execution Schedule Builder module creates a unique execution schedule S from actions in I. Both Best Model Selectors and Execution Schedule Builders can be customized, as well as it is possible to program custom action predicates each having its own customizable *Environment* interface. Moreover, the acthex system features iterative evaluation of P'. The iteration process can be controlled as described in Section 3.

5 Application and Evaluation

acthex can be fruitfully used in a variety of contexts, especially when it is expected to take actions which have impact on actual dynamic environments, and which require to repeatedly take new decisions. In this respect, logic-based games are the ideal testbed: we showcase here two pilot applications (addons) we developed using the acthex system.

Sudoku Addon. This addon allows to maintain a Sudoku table of arbitrary size and to perform operations on it. Sudoku tables are seen as stored within the external environment. The addon provides a single action predicate $\#sudoku[A, O_1, O_2, O_3]\{O, P\}[W : L]$, where A is an operation type and O_1, O_2, O_3 are parameters, depending on the operation type. Possible actions are the insertion of a number into a cell, exclusion of a candidate number from the possible values of a cell, and printing the current table in various formats. Other external predicates allow to query the content of the current table. This addon permitted us to experiment with the incremental application of Sudoku inference rules as described in [3]. Large Sudoku tables cannot be solved by pure guess & check strategies: on the other hand, acthex allows to iterate over partially complete tables, and to repeatedly apply a number of deterministic inference strategies depending on the current resolution progress. Our acthex-based iterative player allows to solve Sudoku tables as large as 81×81, which are far out of the performance reach of an ASP-based system using a pure guess & check strategy[5].

Reversi Addon. The Reversi addon allows for playing an online version of the popular board game Reversi. acthex allows to program Reversi heuristic rules using a logic

[5] Detailed results are available at http://www.kr.tuwien.ac.at/research/systems/dlvhex/actionplugin/SudokuAddon.html#sbench.

program and to perform actual actions depending on the move of choice. Here the environment includes an external web gaming site[6]; we developed Javascript and Perl scripts in order to access and perform actions on the site, and attached them to the execution of the action atom $\#reversi[A, O_1, O_2]\{O, P\}[W : L]$, where A selects an action type and O_1 and O_2 are parameters, depending on the action type. Possible actions are: setting the game number, logging in, making a move, and waiting until the opponent makes their move. Some external predicates are available for retrieving the current status of the game and the corresponding board. The usage workflow of the Reversi addon is straightforward: after initialization, each iteration extracts the current board state from the Web by means of proper external atoms and performs reasoning about the next move in a logic program, using commonly known heuristic rules for Reversi[7]. The chosen move triggers an action which is executed on the game web site. The iteration progress is then suspended by means of a wait action, which will let a further iteration start when the game opponent replies to the last move. The odering of actions is controlled by the precedence feature of acthex, while the end of the game is detected by means of an external atom, causing to end iteration when a game terminates.

6 Conclusion

In this work we have enriched the acthex semantics by new features and provided an implementation on top of the dlvhex reasoner for HEX-programs. Moreover, an *iteration* framework allows for repeating the evaluation of an acthex program and consequent execution of actions. For evaluation, we applied our system to logic games (Sudoku and Reversi) exhibiting scalability to larger instances and modeling strength. Further work is planned, especially concerning evaluation efficiency. For instance, we are currently considering an incremental evaluation approach similar to iclingo [6], although the latter serves a different purpose, since it does neither address action execution nor maintain arbitrary state information, and hence is less expressive (e.g., for re-planning).

References

1. Basol, S., Erdem, O., Fink, M., Ianni, G.: HEX programs with action atoms. In: International Conference on Logic Programming, Technical Communications. pp. 24–33 (2010)
2. Buccafurri, F., Leone, N., Rullo, P.: Strong and weak constraints in disjunctive datalog. In: Logic Programming And Nonmonotonic Reasoning, pp. 2–17. Springer (1997)
3. Calimeri, F., Ianni, G., Perri, S., Zangari, J.: The eternal battle between determinism and nondeterminism: preliminary studies in the sudoku domain. In: RCRA (2013), submitted.
4. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In: IJCAI. pp. 90–96. Professional Book Center (2005)
5. Febbraro, O., Leone, N., Grasso, G., Ricca, F.: Jasp: A framework for integrating answer set programming with java. In: KR (2012)
6. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: Engineering an incremental ASP solver. In: ICLP. pp. 190–205 (2008)

[6] "Your Turn My Turn", available at http://www.yourturnmyturn.com

[7] See e.g. the Strategy guide for Reversi at http://www.samsoft.org.uk/reversi/strategy.htm