ASP Taking Action

Michael Fink
Vienna University of Technology,
Austria

Abstract

The acthex formalism generalizes logic programs under the answer-set semantics, that may have access to external sources (aka HEX-programs), by introducing action atoms in rule heads in order to effect an external environment. In this article we briefly motivate and review the main concepts of the framework. Prospective applications are dynamic in nature and may interleave reasoning with actual action execution. As a representative, we report on an academic challenge where the framework may be fruitfully applied.

Motivation

One hammer never fits all nails. Despite its success as a declarative problem solving approach, not least due to availability of efficient solvers, this certainly also holds for declarative logic programming paradigms such as Answer Set Programming (ASP). Even if it is not a matter of expressivity, efficiency considerations may call for the use of dedicated algorithms (for instance, how about computing physical properties of 2-dimensional objects represented as convex polygons?). Moreover, current trends in computing, such as context awareness or distributed systems, raise the need to access external data sources, e.g., on the Web. Extensions of ASP, such as HEX-programs [7], that incorporate external means of computation and sources of information have been developed precisely for that purpose. They have successfully been exploited for many applications, including querying data and ontologies [6, 8, 9], e-government [12], fuzzy answer set programming [10], multi-context reasoning [5, 2], conditional planning [11].

However, *dynamic systems* impose further challenges. External atoms are purposely stateless (yielding a purely declarative semantics) and not intended to change the state of external sources. While dynamic systems are concerned with states that evolve over time and computation becomes stateful. A common way to couple declarative reasoning tasks with code that actually effects an exogenous environment is roughly to: wrap a declarative solver in a procedural language, execute it iteratively, and at each step parse its answer sets and establish ad-hoc links with effective code. For a less cumbersome and more rigorous interaction, a generalization of HEX-programs called acthex [1, 4] aims at tighter integration under declarative control, essentially by means of so-called action atoms in rule heads. In the following, we briefly review the main concepts of the acthex framework, before we turn to an academic challenge where we expect to fruitfully exploit acthex-programs to improve over an existing HEX-based solution.

Programs with External and Action Atoms

The acthex formalism [1, 4] generalizes HEX programs [7] 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.

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 (input and output) terms. Formally, its semantics is given by an associated boolean function $f_{\&g}$ of arity $n + m + 1$. Intuitively, given an interpretation I (for ordinary atoms) and $n+m$ ground terms, the corresponding ground external atom evaluates to true under I if $\mathrm{t}f_{\&g}$ returns 1.

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. Semantically, action atoms are assigned an $n + 2$-ary action function $f_{\#g}$. It takes an environment state E, an interpretation I, and n ground terms as its input and returns an environment state E', thus intuitively capturing the execution of the corresponding ground action. 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. □

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 (taking into account level and weight associated with action atoms);

(ii) any (best) model originates a set of corresponding *execution schedules* S, i.e., a sequence of actions to execute (taking into account actions' options and preferences);

(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 [7], i.e., using Herbrand interpretations, the grounding of P wrt. the Herbrand universe, and the FLP reduct, where ground action atoms in rule heads are treated like ordinary atoms as far as answer set computation is concerned.

An implementation[1] of the acthex framework has been realized as an extension to the dlvhex system. Compared to the workflow of an answer set solver, it also computes execution schedules and executes one of it according to the semantics of acthex programs given a selection policy of execution schedules (e.g., first computed), and executable code provided for action predicates. A toolkit facilitates the development of libraries of action predicates with some example libraries available.

Several problems of practical importance and dynamic nature fit the realm of potential acthex applications, including knowledge base updates, reactive reasoning over changing data, the interleaving of (re-)planning with actual plan execution, logic-based agent programming, etc. For the remainder of this article, however, let us focus on an interesting academic challenge.

Angry Birds AI Competition – a Challenge

Angry Birds is a very popular video game where the main goal is to shoot at some pigs by means of birds of different characteristics from a slingshot. The game field (which is static until the player moves) features some structures that shelter pigs. Structures can be very complicated and can involve a number of different object categories with different properties, like wood, ice, stone, etc. The game scenario evolves largely complying with physics laws on a bi-dimensional plane; thus, it is possible, in principle, to infer how a structure will change if hit at a certain position by a certain bird.

The Angry Birds AI Competitions[2] are designed to test the abilities of Angry Birds artificial agents, playing on a variety of levels, on the Google Chrome version of the game. The competition runs on a client/server architecture, where the server runs an instance of the game for each participating agent. Each agent runs on a client computer, and communicates with the server according to a given protocol that allows agents to fetch screenshots of their game state at any time. An artificial player can also obtain the current high scores for each level, and can prompt the server for executing a shot, which will in turn be performed in the corresponding game state. The long term goal of the Competition is to foster the building of AI agents that can play any new level better than the best human players. In order to successfully solve this challenge, participants are solicited to combine different areas of AI such as computer vision, knowledge representation and reasoning, planning, heuristic search, and machine learning. Successfully integrating methods from these areas is indeed one of the great challenges of AI.

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

[2] https://aibirds.org/

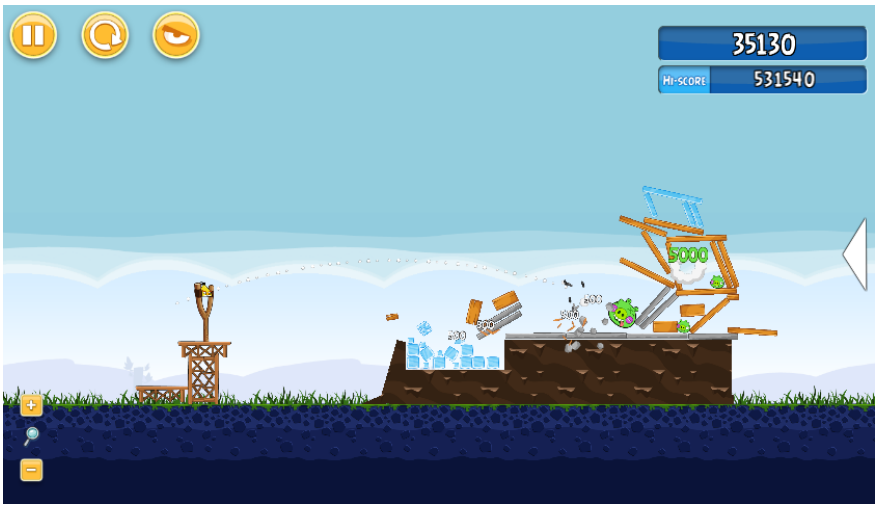

Figure 1: A successful shot featuring an exploding black bird.

In a joint effort of two groups at Technische Universität Wien (TUWIEN) and Università della Calabria (UNICAL) we have developed a participating, logic programming based agent [3]. It is called AngryHEX and realized using HEX-programs. Our agent builds on the Base Framework provided by the organizers and extends it with declarative means for decision making modules. Declarative logic programming kicks in on two different layers for AngryHEX: the *tactics* layer, which plans shots, and decides how to complete a level; and the *strategy* layer, which decides the order of levels to play and possible multiple attempts to solve the same level.

Tactics. The tactics layer is declaratively realized by a HEX-program that computes optimal shots based on information about the current scene and on domain knowledge modeled as part of the program. Its input comprises scene information encoded as a set of logic program facts (position, size and orientation of pigs, ice, wood and stone blocks, slingshot, etc.); its output are answer sets that contain a dedicated atom describing the target to hit, and further information about the required shot. Physics simulation results and other information are accessed via external atoms. For instance, the following rule intuitively represents the likelihood that an object O_2 falls when O_1 is hit, where the fact whether O_1 can push O_2 at all depends on an external (physics) computation taken into account by the external atom $\&canpush$.

$$\begin{aligned} pushDamage(O_2, P_1, P) \quad \leftarrow \quad & pushDamage(O_1, \_, P_1), \\ & P_1 > 0, \&canpush[ngobject](O_1, O_2), \\ & pushability(O_2, P_2), P = P_1 * P_2/100. \end{aligned}$$

Strategy. This layer decides, at the end of each level, which level to play next. This layer is also realized declaratively as an (ordinary) ASP program encoding our strategy on three priority levels: (1) each available level is played once; (2) levels where the agent score differs most from the current best score are selected; (3) levels where AngryHEX achieved a score higher than the current best scores and that have the minimum difference from the best score, are selected. For each level, the strategy layer keeps track of previously achieved scores and previously selected initial target objects.

AngryHEX performed very well in the 2013 Competition at IJCAI, reaching semi-finals. Notably, AngryHEX kept the first place out of 20 participants, in both the two qualification rounds. Benchmarking performed by the Competition Organizer after the Competition also shows AngryHEX ranking with the best score in the first 21 levels of the 'Poached Eggs' level set. The agent also participated in the 2014 Competition at ECAI this year with minor improvements, finishing quarterfinalist and outperforming the 2013 winner (which participated unmodified however).

The competitions revealed several aspects for AngryHEX improvement. A very important issue for its competitivness, which became clear already in 2013 but more obvious this year, is to consider tactics on a more global scale beyond shot by shot analysis. Here is where acthex may kick in: rather than calling dlvhex iteratively for (a single) shot analysis, an acthex-program might suitably represent and encode planning over a sequence of shots taking, e.g., the order and type of available birds into account. Also, interleaving its execution with monitoring and re-planning after performing individual shots might increase its robustness in case of unexpected outcomes. Since it is precisely for capabilites like these that acthex sets out to facilitate it would be a natural next step to evolve AngryHEX into an acthex agent.

Conclusion

The acthex framework is a promising generalization of HEX-programs, i.e., logic programs under the answer-set semantics with access to external sources, aiming at the realization of reasoning tasks for dynamic systems under declarative control. Beyond evaluating acthex in academic settings like the presented Angry Birds AI Competition, an interesting application (and hot topic) would for instance be the implementation of approaches to stream reasoning. In addition to applying the framework as a driving force for further development, also interesting theoretical issues, such as regarding program termination, branching in the environment space (e.g., due to nonmonotonic actions), or incorporating parallel execution schedules remain for future work.

References

[1] Basol, S., Erdem, O., Fink, M., Ianni, G.: HEX Programs with Action Atoms. In: Hermenegildo, M., Schaub, T. (eds.) Technical Communications of the 26th International Conference on Logic Programming (ICLP'10). Leibniz International Proceedings in Informatics (LIPIcs), vol. 7, pp. 24–33. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2010), http://drops.dagstuhl.de/opus/volltexte/2010/2580

[2] Brewka, G., Eiter, T.: Equilibria in Heterogeneous Nonmonotonic Multi-Context Systems. In: Holte, R.C., Howe, A. (eds.) 22nd AAAI Conference on Artificial Intelligence (AAAI'07). pp. 385–390. AAAI Press (2007), http://www.informatik.uni-leipzig.de/~brewka/papers/Equilibria.pdf

[3] Calimeri, F., Fink, M., Germano, S., Ianni, G., Redl, C., Wimmer, A.: Angryhex: an artificial player for angry birds based on declarative knowledge bases. In: Baldoni, M., Chesani, F., Mello, P., Montali, M. (eds.) Proceedings of the Workshop Popularize Artificial Intelligence co-located with the 13th Conference of the Italian Association for Artificial Intelligence (AI*IA 2013), Turin, Italy, December 5,

2013. CEUR Workshop Proceedings, vol. 1107, pp. 29–35. CEUR-WS.org (2013), http://ceur-ws.org/Vol-1107/paper10.pdf

[4] Eiter, T., Fink, M., Ianni, G., Krennwallner, T., Schüller, P.: Pushing Efficient Evaluation of HEX Programs by Modular Decomposition. In: Delgrande, J., Faber, W. (eds.) 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11). LNAI, vol. 6645, pp. 93–106. Springer (May 2011), http://www.kr.tuwien.ac.at/staff/tkren/pub/2011/lpnmr2011-hexdecompeval.pdf

[5] Eiter, T., Fink, M., Schüller, P., Weinzierl, A.: Finding explanations of inconsistency in multi-context systems. Artif. Intell. 216, 233–274 (2014), http://dx.doi.org/10.1016/j.artint.2014.07.008

[6] Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artificial Intelligence 172(12-13), 1495–1539 (2008)

[7] Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In: Kaelbling, L.P., Saffiotti, A. (eds.) 19th International Joint Conference on Artificial Intelligence (IJCAI'05). pp. 90–96. Professional Book Center (2005), http://www.kr.tuwien.ac.at/staff/eiter/et-archive/ijcai05-hex_asp.pdf

[8] Hoehndorf, R., Loebe, F., Kelso, J., Herre, H.: Representing default knowledge in biomedical ontologies: application to the integration of anatomy and phenotype ontologies. BMC Bioinformatics 8, 377 (2007)

[9] Marano, M., Obermeier, P., Polleres, A.: Processing RIF and OWL2RL within DLVHEX. In: Hitzler, P., Lukasiewicz, T. (eds.) 4th International Conference on Web Reasoning and Rule Systems (RR'10). LNCS, vol. 6333, pp. 244–250. Springer (2010)

[10] Nieuwenborgh, D.V., Cock, M.D., Vermeir, D.: Computing fuzzy answer sets using dlvhex. In: Dahl, V., Niemelä, I. (eds.) 23rd International Conference on Logic Programming (ICLP'07). LNCS, vol. 4670, pp. 449–450. Springer (2007)

[11] Nieuwenborgh, D.V., Eiter, T., Vermeir, D.: Conditional planning with external functions. In: Baral, C., Brewka, G., Schlipf, J.S. (eds.) 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07). LNCS, vol. 4483, pp. 214–227. Springer (2007)

[12] Zirtiloğlu, H., Yolum, P.: Ranking semantic information for e-government: complaints management. In: 1st International Workshop on Ontology-supported Business Intelligence (OBI'08). p. 7. No. 5 in OBI'08, ACM (2008)