Contribution to the Panel on Abstractions in AI and Software Engineering

Patrick COUSOT
École Normale Supérieure
45 rue d'Ulm, 75230 Paris cedex 05, France
mailto:Patrick.Cousot@ens.fr, http://www.di.ens.fr/~cousot

SARA'2000, Austin, TX July 27$^{\text{th}}$, 2000

Abstractions in AI and Software Engineering

In the Software Engineering community, abstraction is an approach to managing complexity. Examples of software abstractions procedural abstractions, data abstractions, inheritance, polymorphism and many others. Programming languages provide such abstraction mechanisms in order to diminish the cognitive demands faced by designers of complex systems.

In the Artificial Intelligence community, abstraction is typically viewed as a mechanism for attacking intractable search problems. For example, the performance of a planning algorithm may be improved by using a hierarchy of abstraction spaces. Search algorithms use such abstraction techniques in order to make search control decisions.

Have the two communities missed an opportunity for cross fertilization? For example, do abstraction techniques developed in SE have any application to AI? Do AI techniques for automatically generating abstractions have any application to SE?

Tom Ellman, Assistant Professor
Department of Computer Science
Vassar College
Poughkeepsie, NY 12601

Voice: (845) 437-5991
FAX: (845) 437-7498
Email: ellman@cs.vassar.edu
Web: http://www.cs.vassar.edu/~ellman

Abstraction in ...

- Generalizing the question, we look for criteria to compare abstraction in:
  - Artificial Intelligence;
  - Software Engineering;
  - Model Checking;
  - Program Analysis;

Abstraction in Program Analysis

- Abstraction $\equiv$ Approximation (for problem simplification);
- Mainly used for static safety analysis (more difficult for liveness);
- Property $\underset{\text{Abstraction } \alpha}{\overset{\text{Concretization } \gamma}{\leftrightarrows}}$ Property;
- Abstraction & concretization not computable;
- Huge variety of abstractions (to finite or infinite domains);
- The abstraction must be effective for all programs, so one abstracts to properties which are likely to appear in many programs;
- Program independent abstractions are manually designed;
- Compositional abstractions but successive weaker abstractions avoided (widening proved more precise);
- Supporting theory: abstract interpretation;

Abstraction in Model Checking

- Abstraction $\equiv$ Approximation (for simplification);
- Mainly used for safety checking (liveness if finite);
- Property $\underset{\text{Abstraction } \alpha}{\overset{\text{Concretization } \gamma}{\leftrightarrows}}$ Property;
- Abstraction & concretization (often) computable;
- Abstractions mainly restricted to state to state, boolean or polyhedral abstraction;
- The model-dependent abstraction is specialized for a given model;
- The abstraction is designed by the user;
- Trial and error successive weaker abstractions are common, usually no compositionality;
- Supporting theory: (weak version of) abstract interpretation;

Abstraction in Artificial Intelligence

- Abstraction $\equiv$ Approximation (for simplification);
- Mainly used for search algorithms/theorem proving;
- Object $\xrightarrow{\text{Abstraction } h}$ Object $\xrightarrow{\text{Concretization } \gamma}$ Set of objects [1];
- Abstraction & concretization always computable;
- "Object to object" ("state to state") abstractions;
- The abstraction is specialized for a search algorithm/theorem prover;
- The abstraction is designed by the algorithm/prover designer;
- Successive automatically generated weaker abstractions are common;
- Supporting theory: general program correctness/logical arguments;

[1] The abstraction $\alpha \in \wp(\mathrm{Object}) \mapsto \wp(\mathrm{Object}) \stackrel{\text{def}}{=} \boldsymbol{\lambda} P \cdot \{h(x) \mid x \in P\}$ is left implicit.

Refinement in Software Engineering

- Refinement (concretization) $\equiv$ delayed decisions and decomposition of correctness proofs in structured program design;
- Program $\xrightarrow{\text{Refinement } \rho}$ Program (ρ often relation [2]);
- The refinement is manual, not computable;
- The refinement makes one choice among a loosely defined set of computer representable possibilities;
- The refinement is designed manually while refining;
- Goal directed refinement where goal is program efficiency and correctness;
- Supporting theory: program semantics (predicate transformers);

---

[2] The concretization $\gamma \in \wp(\text{Program}) \mapsto \wp(\text{Set of programs}) \stackrel{\text{def}}{=} \boldsymbol{\lambda} P \cdot \{Q \mid \langle Q, P \rangle \in \rho\}$ is often left implicit.

Abstraction in Software Engineering

- Abstraction $\equiv$ Parameterized program structuration (some part is left partially specified, no approximation, it's generalization);
- Mainly used for program and data parameterized generic program design;
- $\boldsymbol{\lambda}$(program, data).procedural body (program, data)
- The abstraction is manual;
- The abstraction makes one choice among a loosely defined set of possible parameters;
- Composition of abstractions is common;
- Supporting theory: none (general program correctness arguments);

Answers to questions [3]

- AI & Soft. Eng. cross-fertilization: abstraction in AI is approximation (and sometime generalization) whereas abstraction in Soft. Eng. is generalization only (beyond the scope of what is currently feasible in AI);
- Has Soft. Eng. applications in AI: almost nothing is automated in Soft. Eng., so the reciprocal is more probable;
- Do automatically generated abstractions apply to Soft. Eng.: at least in program analysis, the design of a suitable abstraction is as difficult as the design of a formal proof, which is beyond the scope of present-day AI techniques, without human-interaction;

[3] Hopefully provocative enough to start up and stimulate the panel discussion.