A semantics for abstraction

Chiara Ghidini[1] and **Fausto Giunchiglia**[2]

Abstract. The goal of this paper is to propose a model-theoretic formalization of abstraction, where abstraction is modeled as two representations, the ground and the abstract representation, modeling the same phenomenon at different levels of detail. Using the framework of Local Models Semantics, the ground and abstract representations are modeled as two sets of (local) first order models, while the relations holding between them are captured by an appropriate "compatibility relation". The tuning of the compatibility relation allows for the definition of the many different kinds of abstraction.

1 Introduction

Abstraction is a mechanism for representing things in a simplified manner, hopefully capturing their essence. Ideally, one would hope to consider all is relevant and drop all the irrelevant details. Humans use it as a way to deal with the complexity of the world; any representation they construct is a simplified version of the world itself. Humans and computer programs also use abstraction to provide an even more simplified version (an abstract representation) of a previously constructed representation (the so-called ground, or concrete, representation). The main reason is economicity, namely the possibility of concentrating, within a possibly very large representation, only on what is really crucial to the matter under consideration. Abstraction has many important applications in natural language understanding, problem solving and planning, explanation, reasoning by analogy, and so on (see [5] for a detailed discussion). A new application, which, we believe will become very important, is the use of abstraction in the study and discovery of mappings between semantically heterogeneous ontologies [3].

In [5, 6, 4] a theory of abstraction mainly based on proof-theoretic notions was provided. Our goal in this paper is to propose a semantic formalization of the notion of abstraction, which is able to capture exactly the same intuitions underlying the work cited above and to show, at the model theoretic level, some of the properties that characterize abstraction as introduced in [5].

Our formalization is based on the *Local Models Semantics*, as defined in [1] and further refined in [2]. Each representation is modeled as a set of (local) first order models, while the fact that the two representations are related is captured by a *compatibility relation* which links what is true in the two sets of models. The novel technical contributions of the paper are: (i) a formulation of abstraction as a mapping between two languages, and a clear characterization of a class of abstractions operating on the structure of formulae (the, so called, atomic abstraction); (ii) a simple formulation of first order Local Models Semantics as a framework to express relations between first order theories; and (iii) the definition of a semantics of (atomic) abstraction and a first investigation of its properties.

The paper is structured as follows. In Section 2 we provide the basic definitions of abstraction. In Section 3 we provide an extension of Local Models Semantics to the case of first order logic, and in Section 4 we present our semantics of abstraction. Finally, we describe some properties of abstractions and we end with concluding remarks.

2 Abstraction

We follow [5], Where Giunchiglia and Walsh describe informally an abstraction as a mapping between two representations of a problem. The basic definition given in [5] formalizes abstraction as a pair of formal systems plus a mapping between the languages of the two formal systems. This definition was motivated by the interest of the authors in studying the proof-theoretic properties of abstraction and their application to theorem proving and, more generally, to reasoning. However, this definition fails to capture the basic intuition that abstraction, since operating at the level of representations, applies *before* the deductive mechinery of a formal theory (and any inference engine built on top of it) is applied. This is the basic fact which differentiates abstraction from most of the other reasoning mechanisms. We formalize abstraction as a pair of (formal) languages plus a mapping between them.[3]

Definition 2.1 (Abstraction) *Given two languages* L_0 *and* L_1, $abs : L_0 \rightarrow L_1$ *is an* abstraction.

Usually a formal language L is defined by the alphabet, the set of well formed terms and the set of well formed formulae (wffs). To simplify matters we take a language to be a set of wffs. L_0 is the *ground language*, while L_1 is the *abstract language*. abs is the mapping function. We restrict ourselves to abstractions which are total, effective and surjective. We want abs to be total since we want to be able to "translate" any well formed formula of L_1 into a formula of L_2. We want abs to be computable since, from an implementational point of view this is the only interesting case. We want abs to be surjective because we want the abstract representation to be completely "generated" from the ground representation. This simplifies issues and, more importantly, corresponds to how abstraction has actually been used in the (Artificial Intelligence) literature. More formally, totality means that for each symbol $s \in L_0$, $abs(s)$ is defined. Surjectivity means that for each symbol $s' \in L_1$ there is an $s \in L_0$ such that $abs(s) = s'$. Also, since abs is a function (and not a relation) we know that if $abs(s) = s_0$ and $abs(s) = s_1$, then $s_0 = s_1$. For the sake of simplicity we assume that abs preserves the names of variables. The case of abs dropping variables is a straightforward extension.

[1] ITC-Irst, Trento I-38100, Italy. Email: `ghidini@itc.it`

[2] Department of Information and Communication Technology, University of Trento, I-38100, Italy. Email: `fausto@dit.unitn.it`

[3] In this paper we do not address the problem of the different transformations that generate abstraction functions (e.g., abstraction created using relations of the form part-of/whole, class/superclass, selection of attributes, and so on.). Instead, we focus on the definition of the semantics of very general and important class of abstraction functions introduced in [5].

To illustrate a typical use of abstraction let us consider the following example, which is a simplified and modified version of an example originally proposed by Jerry Hobbs in his paper on "*granularity*" [7].

Example 2.1 Assume T_0 is a complex theory of the world containing a large number of agents and objects, where places are points in the Euclidean 3-space. Let L_0 be the first order language used to describe what is true in T_0. For the purpose of this example we assume that L_0 contains: (i) a number of constants $table, chair, lamp, \ldots, b_1, b_2, \ldots$ for objects, where $b_1, b_2, \ldots$ are constants for blocks; (ii) positions, represented as triples $\langle x, y, z\rangle$ of real numbers in a Euclidean 3-space; and (iii) a predicate $at(obj, x, y, z)$ saying that object obj is at position $\langle x, y, z\rangle$.

Suppose we are only concerned with discrete positions in the block world. Our goal is to define an abstract theory T_1, whose language L_1 is able to describe: (i) a small subset of the objects of L_0. In particular only the table $table$, and the blocks $b_1, b_2, \ldots$ that are **on** the table $table$; (ii) positions represented as squares on a 100×100 board, whose lower left corner we will assume to be at the origin; (iii) a predicate $on(obj, x, y)$ similar to the one in L_0, where the vertical dimension is abstracted away; and (iv) an entity EE, for "everything else", used to collapse together all the irrelevant things of L_0.

Following the intuitions presented in [7], and the definition of abstraction presented in [4], the mapping between L_0 and L_1 can be represented using a function $abs : L_0 \to L_1$ defined as follows:

$abs(table) = table,$ for the table $table$.
$abs(b_i) = b_i,$ for all blocks b_i **on** the table $table$.
$abs(x) = EE,$ for all other objects x in L_0.
$abs(\langle x, y, z\rangle) = \langle int(x), int(y)\rangle,$ for $0 \leq x, y \leq 100, z = 0$.
$abs(\langle x', y', z'\rangle) = EE,$ for all other locations $\langle x', y', z'\rangle$ in L_0.
$abs(on(obj, \langle x, y, z\rangle)) = on(obj, \langle x, y\rangle)$.

where for each real number x, $int(x)$ is the greatest natural number n (including 0) such that $n \leq x$.

Following [4], we restrict ourselves to consider L_0 and L_1 as first order languages. Furthermore, adopting a definition given in [4], suitably modified to consider languages and not formal systems, we further restrict ourselves to *atomic abstractions*, namely, to abstractions which map atomic formulae, and keep the logical structure unmodified. Formally:

Definition 2.2 (Atomic Abstraction) $abs : L_0 \to L_1$ *is an* atomic abstraction *iff*

- $abs(\alpha \circ \beta) = abs(\alpha) \circ abs(\beta)$ *for all binary connectives* $\circ$*;*
- $abs(\odot\alpha) = \odot abs(\alpha)$ *for all unary connectives* $\odot$*;*
- $abs(\diamond x.\alpha) = \diamond x.abs(\alpha)$ *for all quantifiers* $\diamond$*.*

Atomic abstractions, as defined in [4], have very nice proof-theoretic properties which make their use in theorem proving very convenient; most noticeably, they increase theoremhood (i.e., they are TI-abstraction in the terminology of [5]) and preserve the shape of proofs. In other words the abstract proof is a simplified version of the ground proof, where all the steps which manipulate the "irrelevant details" are deleted. Our main interest in atomic abstractions in this paper is that, first, they are simpler to handle; second, they are very large class which contains most of the abstractions which can be found in the literature (see [5] for a long list of examples); and, finally, in these abstractions, details are deleted by operating only on the signature (alphabet). This seems the most basic and simplest form of abstraction one could think of. In the following we talk of abstractions meaning atomic abstractions.

Let us now consider a classification of abstraction, given by following the recursive definition of a well formed formula.

1. *Symbol abstractions*. These abstractions operate on symbols and collapse them together. They can operate on constants:
$$c_1, \ldots, c_n \in L_0, c \in L_1 \text{ and } abs(c_i) = c, \text{ for all } i \in [1, n],$$
on functions:
$$f_1, \ldots, f_n \in L_0, f \in L_1 \text{ and } abs(f_i) = f, \text{ for all } i \in [1, n],$$
and on predicates:
$$p_1, \ldots, p_n \in L_0, p \in L_1 \text{ and } abs(p_i) = p, \text{ for all } i \in [1, n].$$
With a liberal extension of Hobbs' proposal which, however, maintains and extends the same intuition, we also call symbol abstractions, *granularity abstractions*;
2. *Arity abstractions*. These abstractions operate on arities and lower them. They operate on function arities:
$$f_1(x_1, \ldots, x_n) \in L_0, f(x_1, \ldots, x_m) \in L_1$$
$$\text{with } n \geq m, \text{ and } abs(f_1) = f,$$
and on predicate arities:
$$p_1(x_1, \ldots, x_n) \in L_0, p(x_1, \ldots, x_m) \in L_1$$
$$\text{with } n \geq m, \text{ and } abs(p_1) = p.$$
3. *Truth abstractions*. These abstractions operate on predicates and map them into the symbol for truth:
$$p(x_1, \ldots, x_n) \in L_0, \text{ and } abs(p(x_1, \ldots, x_n)) = \top.$$

Example 2.1 provides an example of a granularity abstraction on constants. An example of granularity abstraction, operating on functions, collapses the unary functions $walkfrom(loc)$, $drivefrom(loc)$, $flyfrom(loc)$ (which take a location and return a location) into the unary function $gofrom(loc)$, thus abstracting away the details of how one moves from one location to another. An example of granularity abstraction on predicates collapses $CUP(obj)$, $GLASS(obj)$, and $BOTTLE(obj)$ into $CONTAINER(obj)$. Typical arity abstractions, which can be applied to situation calculus, drop the situation argument s thus obtaining, for instance, $ON(obj_1, obj_2)$ from $ON(obj_1, obj_2, s)$. Finally, the most classical example of truth abstraction was introduced in Abstrips [8] to drop supposedly irrelevant preconditions of operators.

Here it is important to notice that the definitions above are quite liberal and many issues are not dealt with. In particular, in most cases it is expected that granularity abstractions operate on functions and predicates of the same arity. Moreover, while merging two functions one may have to choose between two different values returned, for the same input values, by the merged functions and dually, while merging two predicates, one should avoid building an inconsistent theory (for instance by collapsing CUP and $GLASS$ with a knowledge base of the following two facts: $CUP(C_1)$ and $\neg GLASS(C_1)$). Finally, to maintain certain properties (e.g., to preserve the shape of proofs) truth abstractions must be handled with a lot of care. For instance, when using them in Abstrips-like reasoning (this beeing by far their most common use), truth abstractions selectively apply only to ground instances of $p(x_1, \ldots, x_n)$ occurring in preconditions to operators.

A dual way to classify abstraction can be provided. This classification is not based on the recursive definition of well formed formulae, but rather on the definition of terms and atomic formulae. We define a *term abstraction* as an abstraction that operates on term symbols and maps ground terms on abstract terms. Term abstractions contain symbol abstractions on constants and functions, and arity abstractions on function arities. We define a *formula abstraction* as an abstraction that operates on predicate symbols and map ground formulae on abstract formulae. Formula abstractions contain symbol abstractions on predicates, arity abstractions on predicate arities, and truth abstraction. To simplify the presentation, the theory provided below is given in terms of term and formula abstractions. Furthermore we assume that term abstractions on function symbols operate on functions with the same arity and defined over the same domain; and the same for formula abstractions over predicate symbols. Notice that this does not make us loose generality as varying arities and domains can be easily obtained by composing different abstraction functions.

3 Local Models Semantics – first order

Our formalization is based on the Local Models Semantics (LMS) formal framework as originally developed in [1]. LMS is here expanded to accommodate the fact that we are dealing with first order languages. In doing that we take into account some of the features and intuitions presented in [2], where a very general context-based logic, called Distributed First Order Logic (DFOL), is presented.[4] The intuition underlying our definitions is to associate to each of the two languages (ground and abstract language) a set of interpretations (a context, as defined in [1]) and to formalize the abstraction mapping as a *compatibility relation* which defines how meaning is preserved in going from the ground to the abstract representation.

3.1 Local models and models

Let $\{L_0, L_1\}$ be the ground and abstract languages connected by a mapping function abs. Let $\overline{M}_i$ be the class of all models (interpretations) of L_i ($i \in \{0, 1\}$). We call $m \in \overline{M}_i$ a *local model* (of L_i).

Since L_0 and L_1 are first order languages, local models are first order models. Let us briefly recall the basic notions of a first order model. A *model* m for a first order language L is a pair $m = \langle \mathbf{dom}, I\rangle$ where $\mathbf{dom}$ is a non empty set called the *domain* of m and I is called *interpretation* function. As usual the interpretation function assigns a n-ary predicate P to an n-place relation $[P]^I \subseteq \mathbf{dom}^n$, a n-ary function f to an $n+1$-place relation $[f]^I$ over $\mathbf{dom}^{n+1}$, and an individual constant c to some element $[c]^I \in \mathbf{dom}$. An assignment in m is a function a that associates to each individual variable of L an element of $\mathbf{dom}$. The satisfiability relation with respect to an assignment a, in symbols $m \models \phi[a]$, is defined as usual. Given a term (formula) s and a variable x, we adopt the standard notation $[s]^I$ and $[x]^a$ to mean the interpretation of s and the assignment of x. If no confusion arises we drop the square brackets "[" and "]" and simply write s^I and x^a. Also, given an assignment a, and an element $d \in \mathbf{dom}$ we write $a[x := d]$ to denote a new assignment a' such that $y^a = y^{a'}$ for all $y \neq x$, and $x^{a'} = d$.

Let $m_0 = \langle \mathbf{dom}_0, I_0\rangle$ and $m_1 = \langle \mathbf{dom}_1, I_1\rangle$ be two models for L_0 and L_1. Following [2], a *domain relation* r_{01} is a relation

$$r_{01} \subseteq \mathbf{dom}_0 \times \mathbf{dom}_1.$$

In [2], domain relations are used to represent the relations between the interpretation domains of two first order theories. Here they are used to represent the relation between the domains of the ground and abstract models. They are the key mechanism which allows to consider the different domains of the ground and abstract spaces. As we can see from the definition above, domain relations are, in their general form, annotated with the subscripts of the domains they relate. In our case we only need one domain relation r_{01} between $\mathbf{dom}_0$ and $\mathbf{dom}_1$. We can therefore safely drop the indexes. Also, as for the abstraction function, we assume that all domain relations are total and surjective functions. That is, for all $d_1, d_2 \in \mathbf{dom}_1$, if $\langle d, d_1\rangle \in r$ and $\langle d, d_2\rangle \in r$, then $d_1 = d_2$. Therefore together with the usual notation $\langle a, b\rangle \in r$, we will sometimes write $r(a)$ to indicate the element b in the pair above.

Let L_0 and L_1 be two first order languages, and let $\mathbf{dom}_0$ and $\mathbf{dom}_1$ be two domains of interpretation for L_0 and L_1, respectively. From now on we indicate with M_0 a subset of $\overline{M_0}$ that contains only local models $m = \langle \mathbf{dom}_0, I\rangle$ over the domain $\mathbf{dom}_0$ and such that all local models in M_0 agree on the interpretation of terms[5]. Similarly for M_1. Intuitively M_0 (resp. M_1) is a set of local models defined over the same domain of interpretation and such that all local models agree on the interpretations of terms. This means that all the elements in M_0 can only differ on the interpretation of predicates.[6]

Given M_0 and M_1, and a domain relation $r \subseteq \mathbf{dom}_0 \times \mathbf{dom}_1$ we define a *compatibility pair* $\mathbf{c}$ (for $\{L_0, L_1\}$) as a pair

$$\mathbf{c} = \langle \mathbf{c}_0, \mathbf{c}_1\rangle$$

where for each $i \in \{0, 1\}$, $\mathbf{c}_i$ is either a local model m in M_i or the empty set $\emptyset$. Notationally, we call $\mathbf{c}_i$ the i-th element of $\mathbf{c}$.

Given M_0 and M_1, and a domain relation $r \subseteq \mathbf{dom}_0 \times \mathbf{dom}_1$, a *compatibility relation* $\mathbf{C}$ (for $\{L_0, L_1\}$) is a set $\mathbf{C} = \{\mathbf{c}\}$ of compatibility pairs $\mathbf{c}$ defined as above. A *model* is a compatibility relation over r which contains at least a pair and does not contain the pair of empty sets.

Definition 3.1 (Model) *Given M_0 and M_1, and a domain relation $r \subseteq \mathbf{dom}_0 \times \mathbf{dom}_1$, a model (for $\{L_i\}$) is a compatibility relation $\mathbf{C}$ such that $\mathbf{C} \neq \emptyset$ and $\langle \emptyset, \emptyset\rangle \notin \mathbf{C}$.*

The intuition is that a model (of an abstraction function) is a set of pairs of models which are, respectively, a model of the ground and of the abstract representation. The empty set $\emptyset$ intuitively describes an inconsistent representation (i.e., the absence of a model). The two conditions in the definition above eliminate meaningless compatibility relations and pairs, namely totally inconsistent structures. In particular the second condition eliminates the mapping between inconsistent ground and abstract spaces and forces us to consider only pairs that are of the form $\langle m_0, m_1\rangle$, $\langle \emptyset, m_1\rangle$, and $\langle m_0, \emptyset\rangle$.

3.2 Local satisfiability and satisfiability

We can now say what it means for a model to *satisfy* a formula of a language L_i. Let $\models_{cl}$ be the (classical) satisfiability relation between local models and formulae of L_i. Let us call $\models_{cl}$ *local satisfiability*. Since an element of a compatibility pair can be either a model m or the empty set $\emptyset$, we extend, by abuse of notation, the usage of $\models_{cl}$ to the case of $\emptyset$. To maintain the intuition that $\emptyset$ models inconsistent spaces, we say that $\emptyset$ satiafies all the formulae ϕ of a language L_i, and we use the notation $\emptyset \models_{cl} \phi$. Notationally, let us write $i : \phi$ to

[4] We do not use DFOL for two reasons. First, from a presentation perspective we want to maintain the style of Local Models Semantics. Second, DFOL is a very general, powerful logic which is much more complex than we need. As the following will make clear, abstraction allows us to make some simplifying hypotheses which refer to the definitions of model given in [2].

[5] Formally, for each $m_a = \langle \mathbf{dom}_0, I_a\rangle$ and $m_b = \langle \mathbf{dom}_0, I_b\rangle$ in M_0 and for each term t in L_0 we have that $t^{I_a} = t^{I_b}$.

[6] Obviously, with the exception of the equality predicate which has the same standard interpretation for all the first order models.

$$\begin{tikzcd}
x \arrow[r, "abs"] \arrow[d, "a_0"'] & x \arrow[d, "a_1"] \\
d_0 \arrow[r, "r"] & d_1
\end{tikzcd}$$

Figure 1. A pictorial representation of preservation of assignments.

mean ϕ and that ϕ is a formula of L_i. This notation and terminology allows us to keep track of the language we are talking about. That is, it allows to easily distinguish formulae $0 : \phi$ of the ground language, and formulae $1 : \psi$ of the abstract language. Also, from now on, we write a to mean a pair of assignments $\langle a_0, a_1\rangle$ such that a_i is the usual first order assignment for the language L_i. Following [4], we have to impose certain limitations on assignments $a = \langle a_0, a_1\rangle$. In particular we consider here only abstraction functions which preserve names of free and bound variables (or drop them), and preserve substitution instances. Therefore we restrict ourselves to consider the pairs of assignments to the variables of L_0 and L_1 which preserve the assignments to the "same" variable x in the two languages.

Definition 3.2 *Let $\mathbf{C}$ be a model over M_0 and M_1, and $r \subseteq \mathbf{dom}_0 \times \mathbf{dom}_1$. Let a_0 and a_1 be two assignments to the variables of L_0 and L_1 respectively. The pair $a = \langle a_0, a_1\rangle$ is an assignment for $\mathbf{C}$ if for all $x \in L_1, r(x^{a_0}) = x^{a_1}$.*

Definition 3.2 forces us to restrict to assignments which preserve the correspondence between variables in the two languages L_0 and L_1 (see Figure 1). From now on, all assignments a satisfy the condition in Definition 3.2, unless otherwise stated.

Definition 3.3 (Satisfiability) *Let $\mathbf{C}$ be a model, $i : \phi$ a formula, and a an assignment for $\{L_0, L_1\}$. $\mathbf{C}$ satisfies $i : \phi$, under the assignment a, in symbols $\mathbf{C} \models i : \phi[a]$, if for all $\mathbf{c} \in \mathbf{C}$*

$$\mathbf{c}_i \models \phi[a_i]$$

where $\mathbf{c}_i \models \phi[a_i]$ if $e \models_{cl} \phi$, where e is either the local model m or the empty set $\emptyset$ defining the element $\mathbf{c}_i$.

Intuitively: a formula of L_i is satisfied by a model $\mathbf{C}$ if the i-th element of all compatibility pairs satisfy it (under the i-th component of the assignment a). Notice that if $\mathbf{c}_i$ is a local model m then $\mathbf{c}_i \models \phi[a_i]$ can be rewritten as $m \models_{cl} \phi[a_i]$. The interesting case is when $\mathbf{c}_i = \emptyset$. Our definition implies that $\mathbf{c}_i \models \phi[a_i]$ for all formulae ϕ in L_i. As we already said, this captures the intuition that if the i-th element of a compatibility pair models an inconsistent "scenario", then it satisfies all formulae in L_i.

The definition of logical consequence extends the one given in [1] and is not relevant to the study of abstraction presented in this paper. It is therefore omitted for lack of space. The notion of *validity* is the obvious one. A formula $i : \phi$ is *valid* if all models satisfy $i : \phi$.

4 A semantics for abstraction

The key idea is to use domain relations and compatibility relations to model, at a semantic level, the syntactic abstraction relation between terms and formulae of the ground and abstract language.

Definition 4.1 (Satisfiability of term abstractions) *Let abs be a term abstraction between L_0 and L_1. Let $\mathbf{C}$ be a model over M_0, M_1, and $r \subseteq \mathbf{dom}_0 \times \mathbf{dom}_1$. We say that $\mathbf{C}$ satisfies the term abstraction abs if*

- *for all $c_1, \ldots, c_n \in L_0$, $c \in L_1$ such that $abs(c_i) = c$, for all $i \in [1, n]$, $\langle c_i^{I_0}, c^{I_1}\rangle \in r$ for every i in $[1, n]$.*

$$\begin{tikzcd}
c_1 \arrow[rrr, bend left, "abs"] \arrow[d, "I_0"'] & c_2 \arrow[rr, "abs"] \arrow[d, "I_0"] & & c \arrow[d, "I_1"] \\
d_1 \arrow[rrr, bend right, "r"'] & d_2 \arrow[rr, "r"] & & d
\end{tikzcd}$$

Figure 2. Term abstractions and domain relations.

- *for all $f_1, \ldots, f_n \in L_0$, $f \in L_1$ such that $abs(f_i) = f$, for all $i \in [1, n]$,*

$$\textit{if} \left[\begin{array}{c} f_1^{I_0}(d_1, \ldots, d_k) = d_{k_1}, \\ f_2^{I_0}(d_1, \ldots, d_k) = d_{k_2}, \\ \cdots \\ f_n^{I_0}(d_1, \ldots, d_k) = d_{k_n} \end{array}\right] \textit{then} \quad f^{I_1}(r(d_1), \ldots, r(d_k)) = d.$$

  where $d = r(d_{k_1}) = r(d_{k_2}) = \ldots = r(d_{k_n})$
- *for all $f_1(x_1, \ldots, x_n) \in L_0$, $f(x_1, \ldots, x_m) \in L_1$ such that $n \geq m$ and $abs(f_1) = f$*

$$\textit{if } f_1^{I_0}(d_1, \ldots, d_m, \ldots, d_n) = d_{n+1}$$
$$\textit{then } f^{I_1}(r(d_1), \ldots, r(d_m)) = r(d_{n+1})$$

It is easy to see that a model satisfies a term abstraction if the domain relation maps all the ground terms (tuples of terms) into the corresponding abstract terms (tuples of terms). Figure 2 shows the effect of granularity abstractions on individual constants. The fact that c_1 and c_2 are abstracted into the same constant c in L_1 is captured, at the semantic level, by imposing that both the interpretations of c_1 and c_2 in $\mathbf{dom}_0$ are mapped into the interpretation of c in $\mathbf{dom}_1$.

Term abstractions on function symbols work in a similar but slightly different way. In abstracting function names, we collapse functions together. The typical example is the abstraction of two ground functions $+$ and $*$ into a single abstract function $\circ$. These functions are usually defined over the same domain, and a crucial problem arises when we have to decide which value to associate to, say, $a \circ b$. Different uses of abstraction can lead to different choices. A possible choice could be to use the value of one of the ground functions. For instance, one could decide to define the function $\circ$ such that $a \circ b$ is the value of $a + b$ for all a, b. But other choices can be made. For instance one could define the function $\circ$ such that $a \circ b = 1$ if both the values of $a + b$ and $a * b$ are even numbers, and $a \circ b = 0$, otherwise. The only constraint that term abstractions impose here is that the two tuples $\langle a, b, x_1\rangle$ and $\langle a, b, x_2\rangle$ belonging to the interpretation of $+$ and $*$ in the ground language, respectively, are mapped via the domain relation r to a tuple $\langle a, b, x\rangle$ belonging to the interpretation of $\circ$ in the abstract language. This is exactly what Definition 4.1 imposes.[7]

The final part of Definition 4.1 concerns arity abstractions on functions. If a n-ary function f_1 is abstracted into a m-ary function f which simply "forgets" about the "non relevant" arguments $x_{m+1}, \ldots x_n$, then the domain relation r will map all the $n+1$-tuples of $f_1^{I_0}$ into $m + 1$-tuples of f^{I_1} obtained by simply eliminating the "non relevant" elements from the initial tuple and by replacing all the remaining elements d_i with the corresponding element d_i' in the abstract domain.

[7] For the sake of explanation we have assumed in this example that abstraction operates only on function names and does not modify constants or individual elements of the domains. In reality, the constraint that term abstraction imposes is that both tuples $\langle a, b, x_1\rangle$ and $\langle a, b, x_2\rangle$ belonging to the interpretation of $+$ and $*$, respectively, are mapped via the domain relation r to a tuple $\langle a', b', x\rangle$ belonging to the interpretation of $\circ$ where $a' = r(a)$ and $b' = r(b)$.

Definition 4.2 (Satisfiability of formula abstractions) *Let $abs : L_0 \rightarrow L_1$ be a formula abstraction. Let* $\mathbf{C}$ *be a model over M_0, M_1, and $r \subseteq \mathbf{dom}_0 \times \mathbf{dom}_1$. We say that* $\mathbf{C}$ *satisfies the formula abstraction abs if for all compatibility pair $\langle \mathbf{c}_0, \mathbf{c}_1 \rangle$ in* $\mathbf{C}$

- *for all $p_1, \ldots, p_n \in L_0$, $p \in L_1$, such that $abs(p_i) = p$ for all $i \in [1, n]$*

$$\begin{array}{rl} \textit{if} & \mathbf{c}_0 \models p_i(x_1, \ldots x_m)[d_1, \ldots, d_m] \textit{ for some } i \in [1,n] \\ \textit{then} & \mathbf{c}_1 \models p(x_1, \ldots x_m)[r(d_1), \ldots, r(d_m)] \\ \textit{if} & \mathbf{c}_0 \not\models p_i(x_1, \ldots x_m)[d_1, \ldots, d_m] \textit{ for some } i \in [1,n] \\ \textit{then} & \mathbf{c}_1 \not\models p(x_1, \ldots x_m)[r(d_1), \ldots, r(d_m)] \end{array}$$

- *for all $p_1(x_1, \ldots, x_n) \in L_0$, $p(x_1, \ldots, x_m) \in L_1$ such that $n \geq m$ and $abs(p_1) = p$,*

$$\begin{array}{rl} \textit{if} & \mathbf{c}_0 \models p_1(x_1, \ldots x_m, \ldots x_n)[d_1, \ldots, d_m, \ldots d_n] \\ \textit{then} & \mathbf{c}_1 \models p(x_1, \ldots x_m)[r(d_1), \ldots, r(d_m)] \end{array}$$

$$\begin{array}{rl} \textit{if} & \mathbf{c}_0 \not\models p_1(x_1, \ldots x_m, \ldots x_n)[d_1, \ldots, d_m, \ldots d_n] \\ \textit{then} & \mathbf{c}_1 \not\models p(x_1, \ldots x_m)[r(d_1), \ldots, r(d_m)] \end{array}$$

- *for all $p \in L_0$, such that $abs(p) = \top$,*

$$\textit{if } \mathbf{c}_0 \models p(x_1, \ldots x_m)[d_1, \ldots, d_m] \textit{ then } \mathbf{c}_1 \models \top$$

$$\textit{if } \mathbf{c}_0 \not\models p(x_1, \ldots x_m)[d_1, \ldots, d_m] \textit{ then } \mathbf{c}_1 \not\models \top$$

A model satisfies a formula abstraction if the satisfiability of formulae, (and of their negation) is preserved throughout abstraction.

In order to exemplify our definitions we sketch a representation of the scenario described in Example 2.1. For the sake of brevity we omit irrelevant details and concentrate on the definition of an illustrative example.

Example 4.1 Let L_0, L_1 and $abs : L_0 \rightarrow L_1$ be the ones in Example 2.1. Let $\mathbf{dom}_0$ and $\mathbf{dom}_1$ be two domains of interpretation that contain all the constants of L_0 and L_1, respectively and let $r \subseteq \mathbf{dom}_0 \times \mathbf{dom}_1$ be a domain relation that follows directly from abs, that is, a domain relation which satisfy the constraints:

$r(table) = table,$ for the table $table$.
$r(b_i) = b_i,$ for all blocks b_i **on** the table $table$.
$r(x) = EE,$ for all other objects $x \in \mathbf{dom}_0$.
$r(\langle x, y, 0 \rangle) = \langle int(x), int(y) \rangle$ for all positions $\langle x, y, 0 \rangle$ in $\mathbf{dom}_0$ with $0 \leq x, y \leq 100$.
$r(\langle x', y', z' \rangle) = EE,$ for all other locations $\langle x', y', z' \rangle$ in L_0.

Let us now take pairs of local models m_0 and m_1 over $\mathbf{dom}_0$ and $\mathbf{dom}_1$ that interpret each constants c in itself. Let $\mathbf{C}$ be any models over r containing these compatibility pairs. It is easy to see that $\mathbf{C}$ satisfies the granularity abstraction on constants "by construction". Let us now restrict to a $\mathbf{C}$ that satisfies also the granularity abstraction on the predicate symbol on. It is easy to see that if m_0 satisfies the formula $on(b, \langle x, y, z \rangle)$, and the block b is on the table, then m_1 satisfies the formula $on(b, \langle int(x), int(y) \rangle)$.

5 Properties of abstractions

Given a model $\mathbf{C}$ and an abstraction abs, we say that $\mathbf{C}$ satisfies abs if it satisfies all the term and formula abstractions in abs.

Theorem 1 *Given a model* $\mathbf{C}$ *for an abstraction abs, a compatibility pair $\mathbf{c} \in \mathbf{C}$, and a ground formula $0 : \phi$, if $\mathbf{c}_0 \models \phi[a]$, then $\mathbf{c}_1 \models abs(\phi)[a]$.*

The proof is by induction on the structure of ϕ. We first prove the theorem for atomic formulae and their negation. Then we use the inductive hypothesis to prove the theorem for generic formulae.

Theorem 1 represents the fact that satisfiability of formulae "increases" within a compatibility pair. That is, given a compatibility pair $\langle \mathbf{c}_0, \mathbf{c}_1 \rangle$ satisfying an abstraction abs, and a formula ϕ satisfied by $\mathbf{c}_0$, we can be sure that the abstraction ϕ' of ϕ is satisfied by $\mathbf{c}_1$. This theorem is the first model-theoretic counterpart of the property of TI-abstraction studied in [5]. From Theorem 1, we easily obtain:

(a) pairs of the form $\langle \emptyset, m \rangle$ never occur in a model $\mathbf{C}$;
(b) if $\mathbf{C} \models 0 : \phi[a]$, then $\mathbf{C} \models 1 : abs(\phi)[a]$;
(c) if $\mathbf{C} \models 0 : \phi[a]$ for all models $\mathbf{C}$, then $\mathbf{C} \models 1 : abs(\phi)[a]$ for all models $\mathbf{C}$.

A consequence of property (a) is that a model $\mathbf{C}$ for an atomic abstraction is composed of pairs of the form $\langle m, m' \rangle$ and $\langle m, \emptyset \rangle$. This reflects, from the model theoretic point of view, a well known property of TI-abstractions. Since they "increase" theoremhood, they can only "decrease" models. Or, analogously they can abstract consistent set of formulae into consistent or inconsistent set of formulae, but they can never map an inconsistent set of formulae into a consistent one. Properties (b) and (c) generalize the property of Theorem 1. In particular property (c) allows us to say that validity of formulae "increases" within the class of models $\mathbf{C}$. That is, if $0 : \phi$ is a valid formula in the class of models $\mathbf{C}$ for the abstraction abs, then $1 : abs(\phi)$ is also a valid formula in the class of models $\mathbf{C}$.

6 Conclusion

In this paper we have proposed a semantics of abstraction based on the intuition that abstraction is a (very important) technique for representing knowledge in context and of reasoning about it. This is a first step. Ongoing work, omitted here for lack of space, is devoted to the study of composition of abstractions. In particular we have proved that models for a composed abstraction $abs_1 \circ abs_2$ can be obtained by suitable combinations of models for abs_1 and abs_2. Future work has to show that our notions are the semantic counterpart of the proof-theoretic notions provided in [5, 6] (by providing correctness and completeness results). On the more applied side, we plan to use the framework provided in this paper in the study and discovery of mappings between semantically heterogeneous ontologies [3]. On the modeling side this is a first step towards a classification of the many different relations which may exist between two contexts.

REFERENCES

[1] C. Ghidini and F. Giunchiglia. Local models semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence*, 127(2):221–259, April 2001.

[2] C. Ghidini and L. Serafini. Distributed First Order Logics. In *Frontiers Of Combining Systems 2*, Studies in Logic and Computation, pages 121–140. Research Studies Press, 1998.

[3] F. Giunchiglia and P. Shvaiko. Semantic matching. In *Proceedings of the IJCAI-03 Workshop on Ontologies and Distributed Systems (ODS 2003)*, volume 71. CEUR-WS, 2003.

[4] F. Giunchiglia and T. Walsh. Abstract theorem proving: Mapping back. Technical Report 8911-16, IRST, Trento, Italy, 1990.

[5] F. Giunchiglia and T. Walsh. A Theory of Abstraction. *Artificial Intelligence*, 57(2-3):323–390, 1992.

[6] F. Giunchiglia and T. Walsh. The inevitability of inconsistent abstract spaces. *Journal of Automated Reasoning*, 11:23–41, 1993.

[7] J.R. Hobbs. Granularity. In *Proc. of the 9th International Joint Conference on Artificial Intelligence*, pages 432–435, 1985.

[8] E.D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. In *Proc. of the 3rd International Joint conference on Artificial Intelligence*, 1973.