# Intelligent Information Integration as a Constraint Handling Problem

Liviu Badea Doina Tilivea

AI Lab, National Institute for Research and Development in Informatics
8-10 Averescu Blvd., Bucharest, Romania
`badea@ici.ro`

**Abstract.** Intelligent Information Integration ($I^3$) aims at combining heterogeneous and distributed information sources by explicitly representing and reasoning about their content, giving the user the illusion of interacting with a uniform system. In this paper we show how query planning in such a system can be reduced to a constraint handling problem. Conceptually, our approach relies on a generalized abductive reasoning mechanism involving so called partially open predicates, which support a seamless combination of backward (goal-directed) and forward reasoning. The original aspects of our approach consist in the early detection of (plan) inconsistencies using forward propagation of constraints as well as in the seamless interleaving of query planning and execution. Unlike other specialized query planning algorithms, for which domain reasoning and query planning are only loosely coupled, our encoding of source and domain models into Constraint Handling Rules allows us to fully and efficiently exploit existing domain knowledge. The ability to automatically derive source interactions from domain models (ontologies) enhances the flexibility of modeling.

## 1 Introduction and motivation

The integration of hybrid modules, components and software systems, possibly developed by different software providers, is a notoriously difficult task, involving various complex technical issues (such as distribution, different programming languages, environments and even operating systems) as well as conceptual problems (such as different data models, semantic mismatches, etc.).

Solving the conceptual problems requires the development of an explicit (declarative) common knowledge model of the systems to be integrated. Such a model is used (by a so-called *mediator* [Wie92]) not only during the development of the integrated system, but also at query time, when it can be manipulated by a *query planner* to solve problems that could not have been solved by any of the specific information sources alone and might even not have been foreseeable by the system integrator.

The *query planner* of the mediator transforms a user query into a sequence of information-source specific queries that jointly solve the original query. The *query execution* module performs the actual source accesses, while the *result integration* component combines the results obtained from the information sources in a globally consistent result.

In this paper we present an original encoding of *query planning* as a *constraint handling problem*. Conceptually, our approach relies on a generalized abductive reasoning mechanism involving so called *partially open predicates*, which support a seamless combination of backward (goal-directed) and forward reasoning. The original aspect of our approach consists in the early detection of (plan) inconsistencies using forward propagation of constraints. This is especially important since the main purpose of query planning is to prevent as many potential failures as possible at the (early) planning stage, i.e. before actually accessing the information sources. The query planner has been implemented using Constraint Handling Rules (CHR) [Fr98] and used within a complete Information Integration System, called SILK [S].

Unlike other specialized query planning algorithms, for which domain reasoning and query planning are only loosely coupled, our encoding of source and domain models into Constraint Handling Rules allows us to fully and efficiently exploit existing domain knowledge. [1]

CHRs also enable a natural interleaving of *query planning* with *execution*. It would be unrealistic to assume that the query planning stage will be able to construct foolproof plans, i.e. plans that do not fail (especially because of the incomplete knowledge available to the planner). Therefore, we should expect failures and hence provide mechanisms for *replanning* during execution.

As opposed to many other more procedural approaches to information integration (e.g. [GPQ97]), we present an approach to $I^3$ using logic not just for modeling information sources and domains, but also for *reasoning* about them, the main advantage of such an approach being its flexibility. (For efficiency reasons, we use a Constraint Logic Programming environment.)

## 2 Modeling information sources

While the data content of an information source can be represented by a number of source predicates *s*, the user might prefer to have a *uniform* interface to *all* the sources instead of directly querying the sources *s*. In fact, she may not even *want* to be aware of the structure of the information sources. Therefore, the mediator uses a uniform knowledge representation language in terms of so-called *"base predicates"*, which represent the user's perspective on the given domain.

There are two main viewpoints on representing sources and their interactions: *"Global as View"* and *"Local as View"* [Lev00]. In the *Global As View* (*GAV*) approach, interactions between sources have to be described explicitly for every combination of sources that interact. For example

$$s_1(Empl, Department, \ldots) \wedge s_2(Empl, Project, \ldots) \rightarrow proj\_dept(Project, Department). \qquad (1)$$

[1] For example, the Information Manifold [LRO96] uses a specialised query planning algorithm only loosely coupled with the domain reasoning module (CARIN). A tighter integration of domain reasoning with query planning allows us to discover inconsistent plans earlier.

expresses the fact that a particular combination of $s_1$ and $s_2$ has property *proj_dept*(*Project, Department*) (which is a “base predicate”).

The need to consider all relevant combinations of sources has the obvious drawback that adding a new source is complicated, since it involves considering all potential interactions with the other sources. (There can be an exponential number of such interactions.)

On the other hand, in the *Local As View* (*LAV*) approach, the interactions between sources are being handled by the query planner at the mediator level. All that needs to be done when adding a new source is to describe all its relevant properties (in the mediator language, i.e. in terms of base predicates). For example, one would independently describe the sources $s_1$ and $s_2$ using the implications

$s_1$(*Employee, Department,…*) $\rightarrow$ *emp_dept*(*Employee, Department*)
$s_2$(*Employee, Project,…*) $\rightarrow$ *emp_proj*(*Employee, Project*)

and delegate the detection of potential interactions between them to the query planner, which would need domain specific descriptions of the base predicates, such as

*emp_dept*(*Emp,Dept*) $\wedge$ *emp_proj*(*Emp,Proj*) $\rightarrow$ *proj_dept*(*Proj, Dept*). (2)

Note that the LAV approach only shifts the description of source interactions from the source level (1) to the level of base predicates (2). This is not just a simple technical trick, since descriptions of the form (2) would be part of the *domain ontology* (which would be developed once and for all and therefore wouldn't change as new sources are added). In the LAV approach, the description of the sources is easier, since the knowledge engineer only has to describe the properties of the newly added source in terms of base predicates, without having to worry about potential source interactions. This latter task is delegated to the query planner.

The query planner presented in this paper can deal with both GAV and LAV approaches. However, we encourage the use of LAV, since it spares the knowledge engineer the effort of explicitly determining and representing the source interactions.

We start by giving the intuition behind our modeling of source descriptions and their encoding using Constraint Handling Rules (CHR) and Constraint Logic Programming (CLP).

*Constraint Handling Rules* (see [Fr98] for more details) represent a flexible approach to developing user-defined constraint solvers in a declarative language. As opposed to typical constraint solvers, which are black boxes, CHRs represent a 'no-box' approach to CLP.

CHRs can be either *simplification* or *propagation* rules.

A *simplification* rule *Head* <=> *Guard* | *Body* replaces the head constraints by the body provided the guard is true (the *Head* may contain multiple CHR constraint atoms).

*Propagation rules Head* ==> *Guard* | *Body* add the body constraints to the constraint store without deleting the head constraints (whenever the guard is true). A third, hybrid type of rules, *simpagation rules* $Head_1$ \ $Head_2$ <=> *Guard* | *Body* replace $Head_2$ by *Body* (while preserving $Head_1$) if *Guard* is true. (Guards are optional in all types of rules.)

Now, a Local As View source description like $s(X) \rightarrow b_1(X), b_2(X), \ldots$ (3)
is implemented using a CHR *propagation rule* $s(X)$ ==> $b_1(X), b_2(X), \ldots$ (4)

which ensures that every time a query is reduced to a source predicate $s(X)$, the source's properties are automatically propagated (thereby enabling the discovery of potential inconsistencies). Propagation rules however, capture only one direction of the implication (3). Since they are unable to deal with the backward direction of (3), they are inappropriate for regressing a goal in terms of base predicates to a subgoal in terms of the source predicate. This has to be explicitly implemented using *goal regression rules* of the form $b_i(X)$ **:-** $s(X)$. (5)

Unfortunately, this simple approach may loop, since it doesn't distinguish between *goals* (which have to be achieved) and *facts* that are simply propagated forward. We address this problem by separating *goals* (denoted simply as ***G**p*) from *facts*, which are asserted to hold, represented as ***H**p*. (In the implementation, ***H**p* is represented as a CHR constraint, *holds*(*p*), while goals ***G**p* are represented explicitly as *goal*(*p*).)

Propagation rules involve *facts*: $\boldsymbol{H}s(X) ==> \boldsymbol{H}b_1(X), \boldsymbol{H}b_2(X), \ldots$ (6)

while goal regression rules involve *goals*: $\boldsymbol{G}b_i(X) <=> \boldsymbol{G}s(X).$ (7)

The following schema (w.r.t. *p*) takes care of the consistency between goals and facts that simply hold (since goals will be achieved eventually, they also have to hold): $\boldsymbol{G}p ==> \boldsymbol{H}p$ (8)

Note that rule (8) should have a higher priority than (7) (***H**p* should be propagated before the goal is replaced by a subgoal in (7)).

We formalize the above distinction between goals and facts by introducing the notion of *open predicates*. The query planner essentially interleaves two reasoning phases: *goal regression* and *forward propagation* of properties ("saturation"). While goal regression uses the current "closure" (body) of a given base predicate, forward propagation may produce *new instances* of the base predicate (which have to hold for the partial query plan to be consistent). Thus, we need to be able not only to refer to the closure of a given predicate, but also to allow it to be "open".

## 2.1 Open predicates

In Logic Programming, normal predicates are *closed*: their definition is assumed to be complete (Clark completion). On the other hand, abducibles in Abductive Logic Programming (ALP) [KKT98] are *completely open*, i.e. they can have any extension consistent with the integrity constraints. For dealing in a formal manner with forward propagation rules in an abductive framework, we need to allow a generalization of abducibles, namely (partially) *open predicates*.

Unlike normal abducibles, open predicates can have definitions $p \leftarrow Body$, but these are not considered to be complete, since during the problem solving process we can add (by forward propagation) new abductive instances of *p*. The definition of an open predicate therefore only partially constrains its extension.

In our CHR embedding of the abductive procedure we use two types of constraints, ***G**p* and ***H**p*, for each open predicate *p*. While ***H**p* represents facts explicitly propagated (abduced), ***G**p* refers to the *current closure* of the predicate *p* (i.e. the explicit definition of *p together* with the explicitly abduced literals ***H**p*). Thus, informally[2], the extension of *p* is $\boldsymbol{G}p = def(p) \vee \boldsymbol{H}p$.

[2] For lack of space, we defer the formalization of these notions to the full paper.

While propagating ***Hp*** amounts to simply assuming $p$ to hold (abduction), propagating ***Gp*** amounts to trying to prove $p$ either by using its definition *def*($p$), or by reusing an already abduced fact ***Hp***. This distinction ensures that our CHR embedding conforms to the usual '*propertyhood view*' on integrity constraints. In fact, our description rules presented below can be viewed as a generalization of abductive logic programs with integrity constraints interpreted w.r.t. the 'propertyhood view'[3].

**Definition 1.** $M(\Delta)$ is a *generalized stable model* of the abductive logic program $\langle P,A,I\rangle$ for the abductive explanation $\Delta \subseteq A$ iff (1) $M(\Delta)$ is a stable model of $P\cup\Delta$, and (2) $M(\Delta) \models I$.

The distinction between propagating ***Hp*** and ***Gp*** respectively can be explained best using an example. Earning a certain income should propagate the *obligation* of having one's taxes paid (represented by the closure ***G****taxes_paid*). On the other hand, one could imagine a scenario in which the taxes are paid ***H****taxes_paid*, without having the corresponding obligation (goal) ***G****taxes_paid* (for example, as a side-effect of a different goal).

The use of *open predicates* allows mixing *forward propagation* of abduced predicates ***Hp*** with *backward reasoning* using the closures ***Gp***. Forward propagation can be implemented using CHR propagation rules, while backward reasoning involves unfolding predicates with their definitions. The definition *def*(*p,Body*) of a predicate $p$ is obtained by Clark completion of its 'if' definitions. For each such predicate we will have an *unfolding rule* (similar to *p* **:-** *Body*) implemented as a CHR simplification rule: ***Gp*** <=> *def*(*p,Body*) | ***G****Body*,

but also a CHR simpagation rule[4] for matching a goal ***Gp*** with any existing abduced facts ***Hp***: $\boldsymbol{Hp}(X_1) \setminus \boldsymbol{Gp}(X_2)$ <=> $X_1{=}X_2$ ; $X_1{\neq}X_2$, $\boldsymbol{Gp}(X_2)$. (re)

This rule should have a higher priority than the unfolding rule in order to avoid re-achieving an already achieved goal. Note that, for completeness, we are leaving open the possibility of achieving $\boldsymbol{Gp}(X_2)$ using its definition or reusing other abduced facts.

Our treatment of open predicates ***Gp*** = *def*($p$) $\vee$ ***Hp*** is subtly different from the usual method [KKT98] of dealing with (partially) open predicates $p$, where a new predicate name $p'$ (similar to our ***Hp***) is introduced and the clause $p \leftarrow p'$ is added to the definition of $p$: $\{p \leftarrow Def,\ p \leftarrow p'\}$. (*)

The difference is that whenever we refer to ***Gp*** we are implicitly trying to prove $p$, either by using its definition *def*($p$) or by reusing an already abduced fact ***Hp***, *but without allowing such an* ***Hp*** *to be abduced in order to prove* ***Gp*** (whereas in (*) treating $p \leftarrow p'$ as a *program clause* would allow $p'$ to be abduced when trying to prove $p$). This is crucial for ensuring a correct distinction[5] between goals ***Gp*** and abducibles ***Hp*** mentioned above (otherwise we would treat the propagation of goals

[3] The detailed presentation of the semantics is outside the scope of this paper and will be pursued elsewhere. Briefly, the goal regression rules play make up the program $P$, the sources are regarded as (temporary) abducibles $A$, while the forward propagation rules play the role of the ALP integrity constraints $I$. Our notion of integrity constraints is however more general than that used in ALP.

[4] The rule is more complicated in practice, due to implementation details.

[5] This distinction is essential only for *partially* open predicates and not for completely open predicates (abducibles).

and obligations incorrectly). Without making this distinction, *we wouldn't even be able to refer to the current closure of p*.

Besides source descriptions (3), we also have to be able to represent logical constraints between base predicates (for example concept hierarchies, integrity constraints, etc.). The same implementation strategy can be used in this case. For example, an implication of the form $b_1(X,Y) \wedge b_2(Y,Z) \rightarrow b(X,Z)$ can be represented using a CHR propagation rule $\boldsymbol{H}b_1(X,Y), \boldsymbol{H}b_2(Y,Z)$ ==> $\boldsymbol{H}b(X,Z)$ and a goal regression rule $\boldsymbol{G}b(X,Z)$ **<=>** $\boldsymbol{G}b_1(X,Y), \boldsymbol{G}b_2(Y,Z)$.

**Example 1.** Assume that in a software company we want to allocate qualified personnel to a new project while not exceeding a given budget. Assume that the company's information system contains separate databases for each department (in our case ***administrative*** and ***development***, which are regarded as source predicates), as well as a database of ***student***s (part-time employees) who could be employed in the project, in case the overall project budget is exceeded by using full-time personnel (students are assumed to have a lower salary).

The end user of the integrated system (for example the company's manager) would use ***base predicates*** such as: *employee*(*Employee*), *qualification*(*Employee, Qualification*), *salary*(*Employee, Salary*). The following description rules are used to represent ***domain-specific knowledge*** (descriptions for base predicates):

*qualification*(*Empl*, *'C'*) → *programmer*(*Empl*) (e1)

*qualification*(*Empl*, *'Prolog'*) → *programmer*(*Empl*) (e2)

*programmer*(*Empl1*), *non_programmer*(*Empl2*) → *Empl1≠ Empl2* (e3)

The following *description rules* are used to characterize the ***source predicates***:

***administrative***(*Empl*, *Qual*, *Sal*) → *employee*(*Empl*), *qualification*(*Empl*, *Qual*), *salary*(*Empl*, *Sal*), *non_programmer*(*Empl*). (e4)

***development***(*Empl*, *Qual*, *Sal*) → *employee*(*Empl*), *qualification*(*Empl*, *Qual*), *salary*(*Empl*, *Sal*), *Sal > 2000*, *programmer*(*Empl*). (e5)

***student***(*Empl*, *Qual*) → *employee*(*Empl*), *qualification*(*Empl*, *Qual*), *salary*(*Empl*, *Sal*), *Sal = 1000*, *programmer*(*Empl*). (e6)

The above description rules assert that:

- people with qualifications in *C* and *Prolog* are programmers, while programmers and non-programmers are disjoint concepts.
- administrative employees are non-programmers
- employees from the development department are programmers (with a salary over 2000)
- students are also programmers and are assumed to have a fixed salary of 1000.

The user might want to allocate two employees *Empl1* and *Empl2* with qualifications *'C'* and *'Prolog'* such that the sum of their salaries does not exceed 3500 (see continuation of the Example in Section 3.3).

## 3 Query planning with Constraint Handling Rules

In the following, we concentrate on a more detailed description of the representation language for source descriptions and domain models, as well as on their implementation using Constraint Handling Rules.

## 3.1 Base predicates, source predicates and description rules

The content of the various information sources is represented by so-called *source predicates*, which will be described at a declarative level in terms of the so-called *"base predicates"*. More precisely, we distinguish between *content* predicates and *constraint* predicates.

*Content predicates* (denoted in the following with $p$, $q$, possibly with subscripts) are predicates which directly or indirectly represent the *content* of information sources. They can be either *source* predicates or *base* predicates.

*Source predicates* (denoted with *s*) directly represent the content of (part of) an information source (for example, a table in a relational database, or the services provided by the interface of a procedural application). Their definitions (bodies) are not explicit at the mediator level, but can be accessed by querying their associated information source.

*Base predicates* (denoted with *b*) are user-defined predicates for describing the domain, as well as the information content of the sources.

As opposed to content predicates, *constraint predicates* (denoted by *c*) are used to express specific constraints on the content predicate descriptions.

For example, a source *s* containing information about underpaid employees (with a salary below 1000), would be described as:

$$s(Name,\ Salary) \rightarrow\ employee(Name) \wedge salary(Name,\ Salary) \wedge Salary{<}1000.$$

(In the above, *s* is a source predicate, *employee* and *salary* are base predicates, while '<' is a constraint predicate.)

Constraint predicates can be either *internal* (treatable internally by the query engine of the source), or *external* (constraints that can only be verified at the mediator level, for example by the built-in constraint solvers of the host CLP environment). Constraints treatable *internally* by the sources can be *verified* at the source level (by the query engines of the sources), but they are also *propagated* at the mediator (CLP) level. Constraints treatable only *externally* need to be both verified and propagated at the mediator level.

A *complete* description of the source predicates in terms of base predicates is neither possible nor useful, since in general there are too many details of the functioning of the sources that are either unknown to the user, or irrelevant from the point of view of the application. Thus, instead of *complete* (iff) descriptions, we shall specify only approximate (necessary) definitions of the source predicates in terms of base predicates (thus, only the relevant features of the sources will be encoded).

In the following, we use a *uniform* notation for the domain and source **description rules**:

$$\forall \overline{X}.\ \ p_1(\overline{X}_1) \wedge \ldots \quad \rightarrow \quad \exists \overline{Z}.\ \ b_1(\overline{Y}_1) \wedge \ldots \ \ \wedge s_i(\overline{Y}_i) \wedge \ldots \ \ \wedge c_j(\overline{Y}_j) \wedge \ldots \qquad (9)$$

where $\overline{X} = \bigcup_i \overline{X}_i,\ \ \overline{Y} = \bigcup_j \overline{Y}_j,\ \ \overline{Z} = \overline{Y} - \overline{X}$ (with variable tuples viewed as sets).

*Description rules* are necessary definitions of (combinations of) source or base predicates in terms of base predicates and constraints. (*Source descriptions* are special cases of description rules. *Integrity constraints* are description rules with only constraints – typically *'fail'* – in the consequent.[6])

Normally, source descriptions are *coarser grained* than the actual information content of the sources (they abstract away the irrelevant details). However, there are situations in which a source description is *finer grained* than the explicit source content, typically due to (meta-level) knowledge about the content of the source that is not explicitly contained in the source.

**Example 2.** A source of cheap cars may not contain the actual prices of the cars, but we may possess the (meta-level) knowledge that these prices are below 10000:

$s(Model) \rightarrow \exists P.\ car(Model) \wedge price(Model, P) \wedge P<10000.$

Such conceptual descriptions that are finer grained than the source content involve existential variables in the consequent and are, in a sense to be made precise shortly, "nondecomposable".

Occurrences of source predicates in the consequent do not represent updates to the corresponding source. Rather, they are typically used for *conversions* from the format used in sources to the common format used at the mediator level, or more generally for expressing *general constraints* that are encoded in the given source predicate. These need to be *verified* rather than propagated.

**Example 3**. Prices of cars may be stored in a different currency, for which a source *s_exchange* (functioning as a conversion table) is available:

$s(Model, Price) \rightarrow \exists P.\ car(Model) \wedge s\_exchange(Price, P) \wedge price(Model, P).$

Description rules provide a convenient and very expressive way of *describing* sources as well as the domain knowledge.

## 3.2 Implementing description rules in CHR

Due to their flexibility and declarative nature, constraint Handling Rules (CHRs) [Fr98] represent an ideal framework for implementing the reasoning mechanism of the query planner.

A description rule of the form (9) will be encoded in CHR using

- *goal regression rules*: for reducing queries given in terms of base predicates to queries in terms of source predicates, and
- *propagation rules*: for completing (intermediate) descriptions in order to allow the discovery of potential inconsistencies.

An implication of the form $p \rightarrow q_1 \wedge q_2 \wedge \ldots$ is *decomposable* into a conjunction of implications

$(p \rightarrow q_1) \wedge (p \rightarrow q_2) \wedge \ldots$ whenever the consequent includes no existential variables. In the general case, however, we need to Skolemize the description rules (9) before decomposing them (decomposition being necessary for generating goal regression rules):

$$\forall \overline{X}.\ p_1(\overline{X}_1) \wedge \ldots \rightarrow \ldots \wedge sk(k, Z_k, \overline{X}) \wedge \ldots b_1(\overline{Y}_1) \wedge \ldots s_i(\overline{Y}_i) \wedge \ldots c_j(\overline{Y}_j) \wedge \ldots \qquad (10)$$

[6] An integrity constraint of the form $\leftarrow p(X),q(X)$ could be written either as $p(X1),q(X2) \text{ ==> } X1 \neq X2$ or as $p(X),q(X) \text{ ==> } fail$.

where $sk(k, Z_k, \overline{X})$ denotes the fact that the existential variable $Z_k$ depends on the variable tuple $\overline{X}$. Essentially, this is a way of writing the Skolem term $Z_k = f_k(\overline{X})$ using a predicate *sk* instead of a function $f_k$. This simplifies our dealing with such Skolems in our CHR implementation, in which *sk* is a CHR constraint subject to the following propagation rule, which ensures that $f_k(X1)=f_k(X2)$ if $X1=X2$:

$$sk(K,Z1,X),\ sk(K,Z2,X) \mathrel{==>} Z1=Z2. \qquad \text{(sk)}$$

### 3.2.1. Goal regression rules

Now, (10) is decomposable into separate implications

$$\forall \overline{X}.\ \ p_1(\overline{X}_1) \wedge \ldots \quad \rightarrow \quad \ldots \wedge\ sk\,(k, Z_k, \overline{X}) \wedge \ldots \quad \wedge\ b_l(\overline{Y}_l). \qquad (11)$$

for each base predicate $b_l$ occurring in the consequent of (10) (in the consequent of (11) we only keep the Skolems for which $Z_k \in \overline{Y}_l$ ).

(11) can easily be viewed as a goal regression rule for $b_l$:

$$\boldsymbol{G}\,b_l(\overline{Y}_l) \quad \Leftarrow\!\!\Rightarrow \quad \ldots\ , sk\,(k, Z_k, \overline{X}), \ldots\ ,\ \ \boldsymbol{G}\,p_1(\overline{X}_1), \ldots \qquad (12)$$

Since (10) is *not* a sufficient definition for the *source* predicates $s_i$ appearing in its consequent ($s_i$ playing here the role of constraints), we do not generate goal regression rules of the form (11) or (12) for $s_i$. Neither do we produce such rules for the constraints $c_j$.

### 3.2.2 Propagation rules

A set of subgoals in terms of source predicates (induced by backward chaining from the user query using the goal regression rules) may not necessarily be consistent. Applying (10) as forward propagation rules ensures the completion ("saturation") of the (partial) query plan and enables detecting potential conflicts *before* actually accessing the sources.

As base predicates *p* are subject not just to normal backward reasoning, but also to forward propagation, they will be treated as open predicates. The forward propagation rules will thus involve their "open" part ***Hp***:

$$\boldsymbol{H}p_1(\overline{X}_1) \wedge \ldots \mathrel{==>} \ldots, sk(k, Z_k, \overline{X}), \ldots, \boldsymbol{H}b_1(\overline{Y}_1), \ldots, \boldsymbol{G}s_i(\overline{Y}_i), \ldots, \boldsymbol{C}c_j(\overline{Y}_j), \ldots \qquad (13)$$

Note that source predicate occurrences $s_i(\overline{Y}_i)$ in the consequent refer to the "closure" ***G****s*$_i$ of $s_i$ (rather than their open parts ***H****s*$_i$), since sources in the consequent are used as constraints to be *verified*, rather than propagated.

We have already mentioned the fact that we have to distinguish between goals involving a given predicate *p* (which will be treated by a mechanism similar to the normal Prolog backward chaining mechanism) and the instances ***Hp*** of *p,* which trigger forward propagation rules. Operationally speaking, while goals ***Gp*** are "consumed" during goal regression, the fact that *p* holds should persist even after the goal has been achieved, to enable the activation of the forward propagation rules of *p*. Goals *p* will therefore have to propagate ***Hp*** (using rule (8)): ***Gp*** ==> ***Hp*** before applying the goal regression rules for *p:* ***Gp*** **<=>** *body*(*p*).

A predicate *p* for which ***Hp*** is propagated *only* by rule (8) is *closed* (***Gp*** = *body*(*p*)) since all propagated ***Hp*** instances will verify *body*(*p*). Propagating ***Hp*** instances in the consequents of other propagation rules makes *p* an open predicate.

$\boldsymbol{C}c_j$ in (13) represent constraint predicate calls (since our system is implemented in the CLP environment of Sicstus Prolog [Sics], we assume the availability of host constraints solvers for the usual constraint predicates).

Source and domain models are described using rules of the form (9), which are then automatically translated by the system into CHR goal regression and propagation rules (12) and (13). (The additional problem-independent rules (8), (re) and (sk) are used for obtaining the complete CHR encoding of a model.)

## 3.3 Source capabilities and query splitting

Instead of directly executing source predicate calls (by actually accessing the sources), sub-goals $s_i(\overline{Y}_i)$ are delayed to enable their aggregation into more specific sub-queries, thereby transporting less tuples at the mediator level. The goal regression rules for source predicates *s* post constraints of the form ***S****s*, which denote the delayed source call: ***G****s*(*Y*) **<=>** ***S****s*(*Y*).

A *query plan* consists of a number of such source predicate calls $\boldsymbol{S}s_i(\overline{Y}_i)$ as well as additional constraints $\boldsymbol{C}c_j(\overline{Y}_j)$ (propagated by forward rules (13)). Note that both types of constraints, ***Hs*** and ***S****s*, are needed, the latter being "consumed" (deleted) after query execution (so that the source will not be queried again with *s(Y)*). On the other hand, ***Hs*** should persist even after executing the source access, because of potential interactions with constraints that may be propagated later on.

*Information sources* are viewed as *collections of source predicates* that can be accessed via a specialized query interface. (Such information sources can be databases, functional applications, etc.) The query planner reduces a query formulated in terms of base predicates to a query in terms of source predicates and constraints. However, since such a “global” query can involve source predicates from *several* information sources, it will have to be to *split* into sub-queries that can be treated by the separate information sources. Since each information source may have its own query processor, we need to explicitly represent the *capabilities* of these query processors. For example, a specific database interface may not be able to deal with arbitrary joins, may allow only certain types of selections and may also require certain parameters to be inputs (i.e. known at query time). Dataflow issues such as input-output parameters are especially important in the case of procedural applications.

Our *query splitting* algorithm (also implemented in CHR) incrementally selects the source predicate constraints ***S****s* (from the global query plan) that can be dealt with by a given source. In doing this, it separates the constraints that can be treated *internally* by the source's query processor from the ones that need to be treated *externally* (at the level of the mediator). In order to minimize the answer sets of queries (which need to be transferred to the mediator level), the source queries are made as *specific* as possible. This is achieved by delegating to the source's query processor as many constraints as possible. (Only the ones that are not treatable internally, are dealt with externally.)

**Example 1 (continued)** In the following, we present the CHR encoding of Example 1 above, as well as the functioning of the latter as an abductive constraint-based query planner. (We recall that, for simplicity, we have used in the above ***H****p* as a shorthand notation for the CHR constraint *holds(p)* and ***G****p* as an abbreviation for the CHR constraint *goal(p)*. ***C****p* represent constraint predicate calls.)

***H****student*(*Empl*, *Qual*) ==> ***H****employee*(*Empl*), ***H****qualification*(*Empl*, *Qual*), (e6-p)
*sk*(*1,Sal,*[*Empl,Qual*]), ***H****salary*(*Empl*, *Sal*), ***C***(*Sal = 1000*), ***H****programmer*(*Empl*).

***G****employee*(*Empl*) <=> ***G****student*(*Empl*, *Qual*). (e6-g1)
***G****qualification*(*Empl*, *Qual*) <=> ***G****student*(*Empl*, *Qual*). (e6-g2)
***G****salary*(*Empl*, *Sal*) <=> *sk*(*1,Sal,*[*Empl,Qual*]), ***G****student*(*Empl*, *Qual*). (e6-g3)
***G****programmer*(*Empl*) <=> ***G****student*(*Empl*, *Qual*). (e6-g4)

(e6-p) is the forward propagation rule corresponding to the description rule (e6), while (e6-g1)–(e6-g6) are its associated goal regression rules. (The CHR encodings for the rest of the descriptions rules (e1) – (e5) are very similar and are skipped for brevity.) Besides the above problem-specific rules, we also use the general rules (8), (re) and (sk) (in this order, and taking precedence over the problem-specific rules).

The query from Example 1

*query*(*Empl1,Empl2*) **:-** ***G***(( *employee*(*Empl1*), *qualification*(*Empl1,'C'*), *salary*(*Empl1,Sal1*),
*employee(Empl2), qualification(Empl2,'Prolog'), salary(Empl2,Sal2), Sal1+ Sal2<3500* )).

will first trigger the goal reduction rule for *employee*, regressing the goal ***G****employee*(*Empl1*) to ***G administrative***(*Empl1, 'C', Sal1*). The latter would then propagate with (e4) ***H****non_programmer*(*Empl1*)[7] (among others). The next goal, ***G****qualification*(*Empl1, 'C'*), will then propagate with (e1) ***H****programmer*(*Empl1*), which will produce an inconsistency with the previously propagated ***H****non_programmer*(*Empl1*) (using (e3)). This will trigger backtracking to the goal ***G****employee*(*Empl1*), which will now be reduced to the source predicate ***G development***(*Empl1, 'C', Sal1*) (propagating ***H****programmer*(*Empl1*), that is consistent with the next goal ***G****qualification*(*Empl1, 'C'*)). ***H development***(*Empl1, 'C', Sal1*) also propagates (with (e5)) ***C***(*Sal1 > 2000*).

A similar chain of reasoning steps will be performed for the sub-goals involving *Empl2*, leading to the source access ***development***(*Empl2, 'Prolog', Sal2*) and the constraint *Sal2 > 2000*. But unfortunately, the constraint *Sal1+Sal2<3500* is now violated by *Sal1>2000, Sal2>2000*, leading to a backtracking step in which the subgoal ***G****employee*(*Empl2*) is solved by accessing the source ***student***(*Empl2, 'Prolog'*), whose salary *Sal2=1000* is consistent with the constraint *Sal1+Sal2<3500*. All the above reasoning steps belong to the query *planning* phase. The resulting plan (consisting of source accesses and constraints)

*development(Empl1,'C',Sal1), student(Empl2,'Prolog'), Sal2=1000, Sal1+Sal2<3500*

will be split into queries for the separate sources (***development*** and ***student*** being assumed to be tables in different databases). While doing this, the constraints treatable internally by the various sources will be attached to the corresponding source queries. In the above example, the constraint *Sal1+ Sal2<3500* involves variables of two separate sources and will not be directly treatable by any of the sources. However, since *Sal2=1000*, it will be simplified to *Sal1<2500*, which is treatable internally by the first source. Therefore, the execution of the plan above will consist in querying the first source with ***development***(*Empl1, 'C', Sal1*), *Sal1<2500* and the second with ***student***(*Empl2, 'Prolog'*).

[7] To be more precise, ***G administrative***(*Empl1, 'C', Sal1*) propagates with (8) ***Hadministrative***(*Empl1, 'C', Sal1*), which in turn propagates ***H****non_programmer*(*Empl1*) (using the forward propagation rule associated to (e4)).

Note that querying the first source with the whole subplan ***development***(*Empl1, 'C', Sal1*), *Sal1<2500* (instead of eagerly querying it with ***development***(*Empl1, 'C', Sal1*) without bothering to construct plans), will typically transport much less tuples from the sources to the mediator (because *Sal1 < 2500* acts as a filter!).

Planning and execution will be *interleaved* in a seamless manner. For example, the second sub-query ***student***(*Empl2, 'Prolog'*) might fail (at query time we might discover that the database contains no students qualified in *Prolog*). One would then have to backtrack to the first subgoal ***G****employee*(*Empl1*) and obtain it from ***student***(*Empl1, 'C'*), presumably allowing ***G****employee*(*Empl2*) to be solved by using a more expensive permanent employee from ***development***(*Empl2, 'Prolog', Sal2*).

Note that if we knew (in the source description (e6)) that students do not know *'Prolog'*, we could have obtained the correct solution without accessing the source ***student***. This remark shows that source descriptions are very useful for pruning the search space at planning time (before actually querying the sources).

As far as we know, our sophisticated use of constraints for query planning and execution outperforms all existing query planning systems due to the *early detection of inconsistencies* as well as due to the *seamless interleaving of planning with execution*.

## 3.4 Correctness and completeness

Our query answering approach is weakly correct, complete and terminates for a set of acyclic description rules. To explain weak correctness, we need to consider the cases in which the sources may violate the integrity constraints. (This may be due to practical difficulties in maintaining the joint consistency of several distributed information sources, updated by different administrators.) In such a case, we do not simply report a global inconsistency and give up answering queries. Instead, we view integrity constraints as prescriptions on all possible query *answers*, rather than as constraints on the sources themselves (the latter viewpoint being difficult to enforce in practice).

In other words, we avoid testing the joint consistency of the sources (which is computationally very expensive anyway) and impose the integrity constraints only on the query answers.

**Definition 2.** An answer to a query is *weakly correct* iff the associated tuples retrieved from the sources verify the integrity constraints. (I.e. any potential violations of the integrity constraints involve source tuples that are not used in the query answer.)

**Example 4.** Assuming we are dealing with the following integrity constraints on the sources s1, s2 and s3:

***H****s1*(*X1*), ***H****s2*(*X2*) ==> *X1=X2*.
***H****s1*(*X1*), ***H****s3*(*X3*) ==> *X1=X3*.

the query ?- *s1*(*X1*), *s2*(*X2*) may return *X1*=a, *X2*=a, which is a weakly correct answer, since it verifies the first IC. But the answer *X1*=a, *X2*=b is incorrect, since it violates the first IC. Note that *X1*=a, *X2*=a is a weakly correct answer for the query above, even if we have a tuple *s3*(c) that potentially violates the second IC. (However, this inconsistency between *s1* and *s3* is irrelevant from the point of view of the given query and should not affect the answer.)

## 4 Concluding remarks and related work

An exhaustive comparison with other information integration systems is impossible, due to lack of space. Briefly, while database oriented approaches to integration (such as *multi-databases* and *federated databases*) use *fixed* global schemas and are appropriate only if the information sources and users do not change frequently, we deal with *dynamically evolving* schemas (especially if the LAV modeling approach is employed). On the other hand, more *procedural* intelligent information integration approaches, like TSIMMIS [GPQ97] and even some declarative systems like MedLan [AART97] or HERMES [Sub], use explicit query reformulation rules[8], but *without the equivalent of our forward propagation rules* (which allow an early discovery and pruning of inconsistent plans before query execution). Our approach is closer to the more declarative systems like SIMS [AKH96], Information Manifold [LRO96] and Infomaster [DG97].

However, while the Information Manifold (IM) uses a special purpose query planning algorithm, we are using a general Constraint Handling module, which allows an easy development of much more *flexible* intelligent information integration frameworks (the descriptions are declarative, while also allowing efficient reasoning about them). Also, the IM query planning algorithm cannot be easily extended to deal with Global as View descriptions (in which interactions between sources are explicit)[9]. Queries involving source descriptions with existential variables are also not treated completely (see the remark in Section 5 of [DG97b]). Additionally, our use of an optimized constraint propagation mechanism (like the one provided by CHR) may help *discover inconsistencies earlier* than in IM (which checks for consistency *all* possible combinations of sources from the buckets)[10].

On the other hand, IM allows using a description logic (DL) in source descriptions. Such DL descriptions can also be reformulated using our description rules. However, we cannot guarantee *complete* reasoning with such descriptions (on the other hand, *full* CARIN is highly intractable in the worst case). IM also doesn't allow reasoning with integrity constraints. In order to guarantee polynomial complexity of the algorithms for dealing with source capabilities, IM uses a more limited source capabilities language than ours (however, the cases of intractability may not appear in real-world cases).

Our approach is also similar to Infomaster (although few details are available in the published papers [DG97,DG97b]), the main difference being that we are using CHRs for reasoning, while Infomaster uses a model elimination theorem prover. (Although Infomaster was not available for a more detailed comparison, we expect a constraint based approach to be more efficient than a model elimination theorem prover.) Due to its use of *safe iff definitions* for representing sources (and domain knowledge), Infomaster employs *gensym predicates* for emulating unidirectional

[8] Such query templates correspond to our goal regression rules.

[9] The interactions between sources from different "buckets" (in the terminology of [LRO96]) need to be taken into account.

[10] Consistency tests in a sophisticated language like CARIN [LR98] being "lazy" (a complex external consistency verifier for CARIN is invoked).

(necessary) source definitions (which represent the typical case, since only very rarely do we have *complete* source descriptions).

COIN [BG97] also uses a CLP framework (Eclipse) for abductive reasoning and CHRs for implementing integrity constraints. However, integrity constraints can be imposed in COIN *only on source predicates*. Thus, COIN domain knowledge reduces to Prolog reduction rules, which are used *only backwards* (during goal regression). The lack of forward propagation rules involving base predicates (and not just sources) makes the discovery of potential interactions between base predicates (and thus the full use of domain knowledge) impossible. COIN also doesn't have a full-fledged query planner (it doesn't aggregate sub-queries to relations of the same source, thereby having to transport large/huge numbers of tuples from the source to the mediator).

CHR and its disjunctive extension $CHR^{\vee}$ [AS98] have also been used by Abdennadher and Christiansen [AS00] as a platform for integrity constraints and abduction.

However, $CHR^{\vee}$ and SLPs [KTW98] and represent more general architectures (very much like CHRs), while we are concentrating on query planning in the framework of intelligent information integration. (While the criterion for goal suspension in SLPs is the instantiation of variables, in query planning we have a more sophisticated criterion stopping unfolding at source predicates, aggregating them into source queries and instantiating variables by executing these partial queries.) We also propose a higher level knowledge representation formalism based on description rules and use CHRs as a means of combining backward and forward reasoning in this formalism.

We are unaware of any $I^3$ system based on SLPs or CHRs.

In a recent paper [GM02], Grant and Minker present an elegant logic-based approach to data integration. However, Grant and Minker deal with the more difficult problem of answering queries using views (which is more appropriate for a data warehousing approach to integration), while we are more interested in what has been recently called ‘real-time data integration’ (which is more appropriate whenever the sources change frequently so that data warehousing is inapplicable). The state of art on answering queries using views is reviewed in [H01], which compares three algorithms, namely the bucket algorithm [LRO96], inverse rules [DG97a] and the MiniCon algorithm [PL01], showing that the MiniCon outperforms the other two.

Finally, Denecker et al. [D95] use the notion of “open predicates” as a synonym of abducibles, or of what we call “completely open” predicates. We generalize abducibles by allowing open predicates to have partial definitions.

The system presented in this paper (implemented in SICStus CHR [Sics]) is fully operational and has been used in several real-world applications involving the integration of molecular biology and genetics resources (databases, knowledge bases and tools), as well as corporate information systems.

**Acknowledgments.** This work was partially supported by the European Commission IST project SILK (IST-1999-11135) [S]. We are grateful to our project partners IQSOFT (Hungary) and EADS (France) for developing the database wrappers and the Sicstus Prolog interface to them, as well as for interesting discussions on the topics presented in this paper. The SILK toolset developed in the framework of the above-mentioned project is a more complex environment including a mediator (presented in this paper), but also a so-called ‘integrator’ (used for model construction and evolution) [BKS02], as well as various wrappers and a developer interface.

## References

[AART97] Aquilino D., Asirelli P., Renso C., Turini F., MedLan: a Logic-based Mediator Language, IEI Technical Report B4-16, November 1997.

[AC00] Abdennadher S., Christiansen H. An Experimental CLP Platform for Integrity Constraints and Abduction. Proc. FQAS-2000.

[AKH96] Y. Arens, C.A. Knoblock, Chun-Nan Hsu. Query Processing in the SIMS Information Mediator, Advanced Planning Technology, A.Tate (ed) ), AAAI Press, 1996.

[AS98] Abdennadher S., H. Schuetz. $CHR^{V}$: a flexible query language. Proc. FQAS-98.

[BG97] S. Bressan and C.H. Goh. Answering queries in context. In Proceedings of the International Conference on Flexible Query Answering Systems, FQAS-98, Roskilde, 1998.

[BKS02] T. Benko, P. Krauth, P. Szeredi. Application Integration through Logic based Model Evolution. Proc. ICLP-2002.

[D95] M. Denecker. A Terminological Interpretation of (Abductive) Logic Programming. Proc. NMR-95, pp.15-29, 1995.

[DG97b] O.M. Duschka and Michael R. Genesereth. Query Planning in Infomaster, Proc.12th Annual ACM Symposium on Applied Computing, SAC '97, San Jose, February 1997.

[DG97a] O.M. Duschka, M. Genesereth. Answering recursive queries using views. PODS-97.

[DG97] O.M. Duschka, M.R. Genesereth. Infomaster - An Information Integration Tool. Tool. Proc. International Workshop "Intelligent Information Integration", KI-97, Freiburg, 1997.

[Fr98] Fruewirth T. Theory and Practice of Constraint Handling Rules, JLP 37:95-138, 1998.

[GM02] J. Grant, J. Minker. A logic-based approach to data integration. TPLP 2002.

[GPQ97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, J. Widom. The TSIMMIS approach to mediation: Data models and Languages. In Journal of Intelligent Information Systems, 1997.

[H01] Alon Halevy. Answering queries using views: a survey. VLDB Journal 2001.

[KKT98] Kakas A., Kowalski R., Toni F. The role of abduction in logic programming, Handbook of logic in AI and LP 5, OUP 1998, 235-324.

[KTW98] Kowalski R., Toni F., Wetzel G. Executing suspended logic programs, Fundamenta Informaticae 34 (1998), 1-22.

[KTW98] R. Kowalski, F. Toni, and G. Wetzel. Executing suspended logic programs. Fundamenta Informaticae, 34(3):203-224, 1998.

[Lev00] Alon Y. Levy , Logic-Based Techniques, in Data Integration Logic Based Artificial Intelligence, Jack Minker (ed). Kluwer, 2000.

[LRO96] Alon Y. Levy, A. Rajaraman, J.J. Ordille, Querying Heterogeneous Information Sources Using Source. Proc. 22nd VLDB Conference, Bombay, India. 1996.

[PL01] Rachel Pottinger , Alon Y. Halevy , Minicon: A Scalable Algorithm for Answering Queries Using Views VLDB Journal 2001.

[Sics] SICS. *SICStus Prolog Manual*, April 2001.

[Sub] V.S. Subrahmanian et al. HERMES: A heterogeneous reasoning and mediator system. http://www.cs.umd.edu/projects/hermes/overview/paper.

[S] The SILK project (IST-11135). http://www.silk-project.com/

[Wie92] Wiederhold G. Mediators in the architecture of future information systems, IEEE Comp. 25(3) 1992, 38-49.

[WT98] G. Wetzel and F. Toni. Semantic query optimization through abduction and constraint handling. Proc. FQAS-98 Flexible Query Answering Systems, LNAI 1495:366-381, 1998.

[W97] Wetzel G. Using integrity constraints as deletion rules, Proc. DYNAMICS'97, 147-161.