A Rule-Based CQL for 2-Dimensional Tables*

Mohand-Saïd Hacid, Patrick Marcel, and Christophe Rigotti

Laboratoire d'Ingénierie des Systèmes d'Information
INSA Lyon, Bâtiment 501, F-69621 Villeurbanne Cedex
Tél : 72 43 85 88 - Fax : 72 43 87 13
{mohand,patrick,crig}@lisi.insa-lyon.fr

Abstract. We describe the core of a rule-based CQL, devoted to the manipulation of *2-dimensional tabular databases*. The rules provide a simple and declarative way to restructure and query tables, and the constraints allow to define cell contents by formulas over concrete domains. We define a model-theoretic semantics and develop an equivalent fixpoint theory that leads to a naive evaluation procedure.

1 Introduction

Tables are a widely used representation of data. Spreadsheet programs and their tabular data models have become a very popular tool for data presentation and analysis. From a database point of view, the multidimensional tabular representations of data and their manipulations used in On-Line Analytical Processing (OLAP) [6,7] are one of the new challenging technologies.

Because of this growing interest, the theoretical foundations of tabular models and their associated languages appear as a motivating field of research. Algebras have recently been designed [8,4], however to our knowledge no rule-based language has been proposed.

In this paper we introduce the model-theoretic and fixpoint semantics of a rule-based language for 2-dimensional tables. The main problem is to handle simultaneously, at a logical level, three important aspects of tabular data representations and manipulations, namely:

- Definition of cell contents by formulas over concrete domains (e.g., the integers, the real numbers).
- Monovaluation of cells, i.e., the association of a unique cell contents to each pair $\langle$*row name, column name*$\rangle$.
- Schema browsing (i.e., table, row and column names used as data).

This difficulty has been overcome by combining techniques stemming from previous works done in the area of databases and logic programming:

- Integration of constraint solving and logic as in CLP [9] and as in Datalog with constraints [10,12].

* In Proc. of CDB'97 (Constraint Databases and their Applications), LNCS 1191.

– Semantics of "monovaluation" as in Datalog with single-valued data functions [2].
– Higher-order syntax as in Hilog [5] allowing schema browsing as in F-logic [11].

The main contribution of this paper is twofold. Firstly, we define a model-theoretic declarative semantics, that allows a high level specification of tabular data manipulations. Secondly, we give a formal operational semantics that is a basis for the reuse of evaluation methods developed for deductive databases and constraint databases. Although the language can be easily generalized to n-dimensional tables, we focus on 2-dimensional tables for the sake of simplicity.

In the following section we illustrate through examples the most salient features of the language. In Section 3 we give its model-theoretic semantics, and an equivalent fixpoint semantics that leads to a naive evaluation procedure. We conclude in Section 4.

2 Informal Overview

In this section, we first outline our data model, and then we present informally the semantics of the query language.

2.1 A 2-dimensional Tabular Data Model

Cells. In this 2-dimensional tabular data model, data are organized in cells. A cell is identified by a *cell reference*, and is associated with a unique *cell contents*.

A cell reference is of the form T(R,C), where T,R and C are *names*, respectively the table name, the row name and the column name.

Example 1. The cell reference school(kate,group) can be used to identify a cell containing the group name of the student Kate.

Structured names can be built from *atomic names* using the constructor "•".

Example 2. The cells describing the results of Kate in mathematics and physics can be identified by the references school(kate,mark•math) and school(kate,mark•phys).

A cell contents is either a *concrete value* or a *name*. Concrete values are elements of a concrete domain (e.g., real, integer) used for numerical computations.

Associations of cells contents with cells references are represented by *ground atoms*. These *atoms* are of two forms: either T(R,C):N or T(R,C)=V, where T(R,C) is a cell reference, N is a name and V is a concrete value.

Example 3. Ground atoms:
school(kate,mark•math)=15
school(kate, favoriteSubject):math

It should be noted that tables, rows and columns are "reified": they belong to the same domain as cell contents. This technique stems from class reification done in F-logic [11]. It provides symmetric treatment to cell references and cell contents. The only restriction is that concrete values can not be used to form cell references.

Tables. A *2-dimensional table* is a set of ground atoms having a common table name, in which the same reference does not appear more than once to ensure cell monovaluation.

Example 4. A table named *school* describing students' results in mathematics and physics, together with their group membership:
{ school(kate,group):gr1, school(mike,group):gr2
school(kate,mark·math)=15, school(mike,mark·math)=14
school(kate,mark·phys)=12}

A table can be viewed simply as a mapping from cell references to cell contents. This is a logical level, at which a table is independent of any graphical representation (Graphical User Interface level) and any storage structure (physical level). Figure 1 shows several graphical presentations of the (logical) table of Example 4. It should also be noticed that in this model an empty cell at the physical level has no direct logical representation.

school	*mark math*	*mark phys*	*group*
kate	15	12	gr1
mike	14		gr2

school	*mark math*	*group*	*mark phys*
mike	14	gr2	
kate	15	gr1	12

school	*mark*		*group*
	math	*phys*	
kate	15	12	gr1
mike	14		gr2

Fig. 1. Several representations of table *school*

Database. A *2-dimensional tabular database* is a set of ground atoms in which the same reference does not appear more than once.

2.2 The Rule-Based Language

We adopt the following conventions: uppercases denote variables and lowercases denote constants.

In this subsection we consider the table *school* of Figure 2.

school	*mark*		*group*
	math	*phys*	
kate	15	12	gr1
mike	14		gr2
john	10	12	gr1
alan	15	16	gr2
suzy	12	11	gr1

Fig. 2. More students

Querying the Database. In its basic form, a query is a conjunction of (possibly non-ground) atoms.

Example 5. The query *"What are the names and the marks in mathematics of students in group gr2 ?"* can be expressed as

?- school(N,mark·math)=X, school(N,group):gr2.

and gives the following answer:

{{ N=mike, X=14 }, { N=alan, X=15 }}

It should be noted that a higher-order syntax stemming from Hilog [5], allows variables to range over atomic names used in column, row and table names. This provides for some kind of schema browsing abilities in the spirit of F-logic [11].

Selection criterion over a concrete domain can be expressed in a constraint language.

Example 6. The query *"What are the names and the marks in mathematics for the students whose mark is between 10 and 13 ?"* can be expressed as

?- school(N,mark·math)=X // X$\geq$ 10 $\wedge$ X$\leq$ 13.

and gives the following answer:

{{ N=john, X=10 }, { N=suzy, X=12 }}

Intensionally Defined Cells. Rules à la Datalog are used to define new cell references (and their associated contents) from existing cells. Concrete values can be specified by means of formulas of the constraint language.

Example 7. Definition of a column in table *school* which gives the average for each student:

school(N,mark·avg)= Z $\leftarrow$ school(N,mark·math)= X,
school(N,mark·phys)= Y // Z = (X+Y)/2.

A representation of the corresponding cells is depicted in Figure 3.

The rules can also be used to define restructurations of data.

Example 8. Restructuration of the data in table *school*, by grouping students along their group names:

school	*mark*			*group*
	math	*phys*	*avg*	
kate	15	12	13.5	gr1
mike	14			gr2
john	10	12	11	gr1
alan	15	16	15.5	gr2
suzy	12	11	11.5	gr1

Fig. 3. Computing averages

schoolByGroup(G·N,mark·M)= X ← school(N,mark·M)= X,
school(N,group):G.

A representation of table *schoolByGroup* is depicted in Figure 4.

schoolByGroup		*mark*		
		math	*phys*	*avg*
gr1	*kate*	15	12	13.5
	john	10	12	11
	suzy	12	11	11.5
gr2	*mike*	14		
	alan	15	16	15.5

Fig. 4. A restructuration

We insist on the fact that this rule only specifies a logical grouping of the students using the contents of cells to form row names. The graphical positions of the rows in Figure 4 is a choice made at the Graphical User Interface level.

Other structurations can be obtained using cell contents to build table names in order to restructure data in several tables.

Example 9. Spliting the table *school* into two tables according to the student groups:
group·G(N,mark·M)= X ← school(N,mark·M)= X,
school(N,group):G.

A representation of the corresponding tables are given Figure 5.

3 Syntax and Semantics

In this section, we formally present the syntax and the declarative and operational semantics of the language. Proofs of the results are given in the appendix.

$group \cdot gr1$	$mark$		
	$math$	$phys$	avg
$kate$	15	12	13.5
$john$	10	12	11
$suzy$	12	11	11.5

$group \cdot gr2$	$mark$		
	$math$	$phys$	avg
$mike$	14		
$alan$	15	16	15.5

Fig. 5. A table for each group

3.1 Syntax

Let $\mathcal{D}_{an}$ and $\mathcal{D}_{cv}$ be two disjoint sets of constants. $\mathcal{D}_{an}$ is the set of *atomic names* and $\mathcal{D}_{cv}$ is the set of *concrete values*. Let $\mathcal{V}_{an}$ and $\mathcal{V}_{cv}$ be two disjoint countable sets of variables, disjoint from $\mathcal{D}_{an}$ and $\mathcal{D}_{cv}$.

Constraint Language. Let $\mathcal{L}_{cv}$ be a constraint language, with a particular set of connectives and quantifiers to build formulas. The constraint language must fulfill the following requirements:

- the set of constants and variables used in $\mathcal{L}_{cv}$ are respectively $\mathcal{D}_{cv}$ and $\mathcal{V}_{cv}$,
- $\mathcal{L}_{cv}$ is closed under substitutions of variables by constants,
- the satisfiability of $\mathcal{L}_{cv}$ is decidable,
- $\mathcal{L}_{cv}$ contains at-least one satisfiable constraint, noted $true$.

In the following, formulas of $\mathcal{L}_{cv}$ are simply called *constraints*.

Example 10. Choosing for $\mathcal{L}_{cv}$ the conjunctions of linear equalities and inequalities (strict or unrestricted) is sufficient to express the examples of Section 2.

Rule-Based Language. We now define the syntactical expressions allowed in the rule-based language:

$$
\begin{aligned}
&name := d_{an} \mid v_{an} \mid name \cdot name \\
&value := d_{cv} \mid v_{cv} \\
&reference := name(name, name) \\
&atom := reference : name \mid reference = value \\
&body := atom, body \mid \epsilon \\
&head := atom \\
&rule := head \leftarrow\ body /\!/\ constraint
\end{aligned}
$$

where $d_{an} \in \mathcal{D}_{an}, v_{an} \in \mathcal{V}_{an}, d_{cv} \in \mathcal{D}_{cv}, v_{cv} \in \mathcal{V}_{cv}$ and $constraint \in \mathcal{L}_{cv}$. If the constraint part is reduced to the constraint $true$ then the rule may be abbreviated $head \leftarrow\ body$.

Let var_{an} be a computable function that assigns to each syntactical expression a subset of $\mathcal{V}_{an}$, corresponding to the set of variables occuring in the expression. If $E_1, \ldots, E_n$ are syntactical expressions, we use $var_{an}(E_1, \ldots, E_n)$ as an abbreviation for $var_{an}(E_1) \cup \ldots \cup var_{an}(E_n)$. The function var_{cv} is defined in an analogous way for the variables in $\mathcal{V}_{cv}$. Let $var = var_{an} \cup var_{cv}$.

A ground atom is an atom A for which $var(A) = \emptyset$. In the following, we will note a rule by $A \leftarrow B_1, \ldots, B_n /\!\!/ C$, where $A, B_1, \ldots, B_n$ are atoms, and $C \in \mathcal{L}_{cv}$. A ground rule is a rule cl for which $var(cl) = \emptyset$.

Range Restricted Rule. A range restricted rule is a rule $cl = A \leftarrow B_1, \ldots, B_n /\!\!/ C$ where $var_{an}(A) \subseteq var_{an}(B_1, \ldots, B_n)$.

Program. A program is a finite set of range restricted rules.

3.2 Semantics

In this section, we give a declarative model-theoretic semantics and an equivalent fixpoint-based operational semantics. This presentation is inspired by the presentation of Datalog made in [3].

Model-Theoretic Semantics.

We note $ref(A)$ the reference part of an atom A.

Consistency and Interpretation. A set I of ground atoms is consistent if $\forall A_1, A_2 \in I, ref(A_1) = ref(A_2) \implies A_1 = A_2$, where "=" is the syntactical equality. An interpretation is a consistent set of ground atoms.

This consistency criterion is drawn from the semantics of Datalog with *single-valued data functions* [2].

Valuation. A valuation ν is a function $\nu = \nu_{an} \cup \nu_{cv}$, where ν_{an} is a total function from $\mathcal{V}_{an}$ into $\mathcal{D}_{an}$, and ν_{cv} is a total function from $\mathcal{V}_{cv}$ into $\mathcal{D}_{cv}$. ν is extended to be the identity on $\mathcal{D}_{an}$ and $\mathcal{D}_{cv}$, and it is extended in a straightforward manner to map atoms to ground atoms.

We also extend ν to be defined on $\mathcal{L}_{cv}$, requiring that ν valuates only the free occurences of the variables in the constraints.

Finally, ν is extended to map rules to ground rules.

Rule Satisfaction. Let cl be a rule $A \leftarrow B_1, \ldots, B_n /\!\!/ C$, and I an interpretation. I satisfies cl, denoted $I \models cl$ iff for each valuation ν, if $\forall B_i, i \in [1, \ldots, n], \nu(B_i) \in I$, and $\nu(C)$ is satisfiable, then $\nu(A) \in I$.

Model of a Program. An interpretation I is a model of a program P denoted $I \models P$, iff $\forall cl \in P, I \models cl$.

It should be noticed that even simple programs may have no model, as it is the case in other languages that allow some kind of monovaluation (e.g., Datalog with single-valued data functions [2], COL [1]).

Example 11. The following program defines two different cell contents for the same cell reference and thus it has no model:

a(b,c):e $\leftarrow$.

a(b,c):d $\leftarrow$.

The same kind of inconsistency can also arise from the use of constraints.

Example 12. The following program has no model, since it associates infinitely many cell contents (all X greater than 0) with the same cell reference:

a(b,c)=X $\leftarrow$ // $X \geq 0$.

We insist on the fact that the valuations map variables of $\mathcal{V}_{an}$ only to atomic names. They don't map variables of $\mathcal{V}_{an}$ to names constructed with "•". This guarantees that if a program admits a model then it admits also a finite model. This is shown in the following example and stated by Theorem 14.

Example 13. Consider the following program:

a(b,c):e $\leftarrow$.

a(X•b,c):e $\leftarrow$ a(X,c):e.

{a(b,c) : e, a(b • b,c) : e} is a finite model of the program since no valuation can map X to b•b. The infinite interpretation {a(b,c) : e, a(b • b,c) : e, a(b • b • b,c) : e, a(b • b • b • b,c) : e, ...} is also a model of this program, but not a minimal one.

Theorem 14. *Let P be a program and I be an interpretation. If P admits a model including I, then P admits a finite model including I.*

Semantics of a Program. The semantics of a program P is given with respect to an interpretation called the input of P. For a program P and an input I, the semantics of P on I is the unique minimal model of P containing I, if it exists. It is denoted $P(I)$.

Theorem 15. *Let P be a program and I be an interpretation. If P admits a model including I, then $P(I)$exists and is finite.*

Fixpoint Semantics.

Immediate Consequence Operator. Let P be a program and I an interpretation. A ground atom F is an immediate consequence for I and P if either $F \in I$, or $\exists cl \in P, cl = A \leftarrow\ B_1, \ldots, B_n /\!\!/ C$ and there exists a valuation ν such that:

- $F = \nu(A)$, and
- $\forall B_i, i \in [1, \ldots, n], \nu(B_i) \in I$, and
- $\nu(C)$ is satisfiable.

For a program P, we define the immediate consequence operator T_P to be a partial mapping from interpretations of P to interpretations of P, such that, for an interpretation I:

$$T_P(I) = \{A \mid A \text{ is an immediate consequence for } I \text{ and } P\},$$

if this set is consistent; otherwise, $T_P(I)$ is undefined. We have the following properties:

Lemma 16 (monotonicity). *Let I, J be two interpretations such that $I \subseteq J$, and $T_P(J)$ is defined. Then $T_P(I)$ is defined and is included in $T_P(J)$.*

Lemma 17. *I is a model of P iff $T_P(I) \subseteq I$.*

Lemma 18. *Each fixpoint of T_P is a model for P.*

Theorem 19. *Let P be a program and I an input such that $P(I)$exists, then T_P has a unique minimal fixpoint that equals $P(I)$.*

Naive Evaluation Procedure.

Let P be a program and I an interpretation, then

- $T_P^0(I) = I$,
- $T_P^{n+1}(I) = T_P(T_P^n(I))$, if defined,

This leads to the following definition of the semantics of a program P on input I:

Theorem 20. *Let P be a program and I an input such that $P(I)$exists. Then the sequence $\{T_P^i(I)\}_i$ reaches a fixpoint after a finite number N of steps, with $T_P^N(I) = P(I)$.*

Suppose that the following property holds: for any constraint C and any variable X, if X is determinated (grounded) by C then we can compute a solution form equivalent to the projection of C on X.

In this case, $T_P(I)$ (if defined) can be computed for any input I, and then Theorem 20 provides a straightforward naive evaluation procedure.

4 Conclusion

We proposed a language that extends the family of rule-based CQL to 2-dimensional tabular data representations and manipulations. We illustrated its use through typical examples. We defined its model-theoretic semantics, an equivalent fixpoint semantics, and gave a naive evaluation procedure. This language provides a declarative way to specify tabular data manipulations. Its operational semantics can serve as a basis for the reuse of evaluation techniques proposed for deductive and constraint databases.

Because of the growing interest in tabular models (e.g., OLAP [6]) the corresponding theoretical basis are currently investigated. Gyssens et al. [8] proposed a model of tabular database and an algebra for querying and restructuring it.

Agrawal et al. [4] defined an algebra for providing On Line Analytical Processing capabilities on top of relational database systems. To our knowledge our work is the first one which proposes a formal rule-based language that handles the following salient features of tabular data representations and manipulations: (1) definition of cell contents by formulas over concrete domains, (2) monovaluation of cells and (3) schema browsing.

This work can easily be generalized to handle n-dimensional tables. We are currently working on the incorporation of aggregate functions to define table summarization.

References

1. S. Abiteboul and S. Grumbach. A rule-based language with functions and sets. *ACM Transactions on Database Systems*, 16(1):1–30, March 1991.
2. S. Abiteboul and R. Hull. Data functions, datalog and negation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 143–153, Chicago, Illinois, USA, June 1988. ACM press.
3. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.
4. R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. Research report, IBM Almaden research center, 650 Harry road, San Jose, CA 95120, 1996.
5. W. Chen, M. Kifer, and D.S. Warren. HiLog: a foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.
6. E. F. Codd, S. B. Codd, and C. T. Salley. Providing olap (on-line analytical processing) to user-analysts: An IT mandate. White paper - URL:http://www.arborsoft.com/papers/coddTOC.html, 1993.
7. R. Finkelstein. Understanding the need for on-line analytical servers. White paper - URL:http://www.arborsoft.com/papers/finkTOC.html, 1995.
8. M. Gyssens, L. V. S. Lakshmanan, and I. N. Subramanian. Tables as a paradigm for querying and restructuring. In *Proceedings of the Fifteenth ACM Symposium on Principles of Database Systems*, Montreal, PQ, Canada, June 1996. ACM press.
9. J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the Fourteenth ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, Germany, January 1987.
10. P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query langages. In *Proceedings of the Ninth ACM Symposium on Principles of Database Systems*, pages 299–313, Nashville, Tennessee, USA, April 1990. ACM press.
11. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, July 1995.
12. P. Z. Revesz. A closed form for datalog queries with integer order. In S. Abiteboul and P. C. Kanellakis, editors, *Proceedings of the Third International Conference on Database Theory*, volume 470 of *Lecture Notes in Computer Science*, pages 187–201, Paris, France, December 1990. Springer-Verlag.

Appendix

This appendix presents proofs of the results stated in Section 3. We follow and extend the proofs given in [3, Chapter 12].

Proof of Theorem 14

For a program P and an input I, let $adom_{an}(P, I)$ be the set of constants from $\mathcal{D}_{an}$ in P and I. Let $maxpoint$ be the maximal number of occurences of the constructor "$\cdot$" in any given $name$ used in P and I. Then we define $Ref(P, I)$ to be the set of all $reference$ expressions constructed from constants in $adom_{an}(P, I)$ using at-most $maxpoint$ occurences of "$\cdot$". As both P and I are finite, so is $Ref(P, I)$.

Now let P be a program and I an input such that P admits a model M including I. Let $M' = \{A \mid A \in M \text{ and } ref(A) \in Ref(P, I)\}$. $Ref(P, I)$ is finite, so M' contains a finite number of $references$. And M' is consistent since M is consistent. Thus M' is finite.

Moreover, M' contains I, since $\forall A \in I, ref(A) \in Ref(P, I)$ and $I \subseteq M$.

We next show that M' is a model of P. Let $cl = A \leftarrow B_1, \dots, B_n /\!\!/ C$ be a rule in P, and ν be a valuation such that $\nu(B_1), \dots, \nu(B_n) \in M'$ and $\nu(C)$ satisfiable. Since $\nu(B_1), \dots, \nu(B_n) \in M'$, each atomic name appearing in $\nu(B_1), \dots, \nu(B_n)$ belongs to $adom_{an}(P, I)$. As cl is range-restricted, each constant in $\nu(A)$ also belongs to $adom_{an}(P, I)$, and hence $ref(\nu(A)) \in Ref(P, I)$. Moreover, since $M \models cl$, and $\nu(B_1), \dots, \nu(B_n) \in M$ and $\nu(C)$ satisfiable, we have $\nu(A) \in M$. As $ref(\nu(A)) \in Ref(P, I)$, $\nu(A) \in M'$. This means that M' is a model of P.

Whence, there exists a finite model of P containing I.

Proof of Theorem 15

Let P be a program and I an interpretation such that P admits a model including I. We define $\mathcal{X}$ to be the set of all models of P including I. $\mathcal{X}$ is non-empty, so we define $J = \cap\mathcal{X}$. By lemma 14 J is finite.

We first show that $J \models P$. Let $A \leftarrow B_1, \dots, B_n /\!\!/ C$ be a rule in P and ν a valuation such that $\nu(B_1), \dots, \nu(B_n) \in J$ and $\nu(C)$ satisfiable. $\forall K \in \mathcal{X}$, since $J \subseteq K$, we have $\nu(B_1), \dots, \nu(B_n) \in K$. Thus, $\nu(A) \in K$, because K is a model of P. This holds for all $K \in \mathcal{X}$. Hence, $\nu(A) \in J$. So J satisfies any rule in P.

Moreover, J includes I, because I is included in all $K \in \mathcal{X}$. By construction, J is the minimal model of P including I. Then $P(I) = J$.

Proof of Lemma 16

Let I, J be two interpretations such that $I \subseteq J$, and $T_P(J)$ is defined. Let A be an immediate consequence for I and P. We show that $A \in T_P(J)$. Since A is an immediate consequence for I and P, at-least one of the two following cases applies:

- $A \in I$. Then $A \in J$, and thus $A \in T_P(J)$,
- there exists a rule $A \leftarrow B_1, \dots, B_n /\!\!/ C$ in P and a valuation ν such that $\forall B_i, i \in [1, \dots, n], \nu(B_i) \in I$ and C satisfiable. As $I \subseteq J$, we have $\forall B_i, i \in [1, \dots, n], \nu(B_i) \in J$, and thus $A \in T_P(J)$.

Since $T_P(J)$ is defined, $T_P(J)$ is consistent. So the set of all immediate consequences for I and P is also consistent. Then $T_P(I)$ is defined with $T_P(I) \subseteq T_P(J)$.

Proof of Lemma 17

Let I be an interpretation and P be a program.

We first prove $I \models P \implies T_P(I) \subseteq I$.

We define $conseq(P, I) = \{A \mid A \text{ is immediate consequence for } I \text{ and } P\}$. Given any A in $conseq(P, I)$, at-least one of the two cases applies:

- $A \in I$, by definition of the immediate consequence.
- there exists a rule $cl \in P$ noted $A' \leftarrow B_1, \ldots, B_n /\!\!/ C$ and a valuation ν such that $\forall B_i, i \in [1, \ldots, n], \nu(B_i) \in I$, C satisfiable, and $\nu(A') = A$. Since I is a model of P, then $I \models cl$, and we have $A \in I$.

Thus $conseq(P, I) \subseteq I$. Moreover, as I is consistent, $conseq(P, I)$ is also consistent. Hence $T_P(I)$ is defined by $T_P(I) = conseq(P, I)$; and we have $T_P(I) \subseteq I$.

We now prove $T_P(I) \subseteq I \implies I \models P$.

Given cl any rule $A \leftarrow B_1, \ldots, B_n /\!\!/ C$ in P, and any valuation ν, if $\nu(B_1), \ldots, \nu(B_n) \in I$ and $\nu(C)$ satisfiable, then $\nu(A) \in T_P(I)$. So $\nu(A) \in I$, because $T_P(I) \subseteq I$, and then $I \models cl$.

Hence, $I \models P$.

Proof of Lemma 18

We recall that I is a fixpoint of an operator T if $T(I) = I$.

Let I be a fixpoint of T_P. That is, for the program P, $T_P(I) = I$. By lemma 17, we have:

$$T_P(I) = I \implies I \models P.$$

Proof of Theorem 19

Let P be a program and I an input such that P admits a unique minimal model $P(I)$including I. We first show that $P(I)$is a fixpoint of T_P:

- $T_P(P(I)) \subseteq P(I)$ because $P(I)$is a model of P;
- we have $T_P(P(I)) \subseteq P(I)$. $T_P(P(I))$ is defined, then by lemma 16 $T_P(T_P(P(I)))$ is also defined, and $T_P(T_P(P(I))) \subseteq T_P(P(I))$. Thus by lemma 17 $T_P(P(I))$ is also a model of P. As $P(I)$is the minimal model of P, $P(I) \subseteq T_P(P(I))$.

Hence $P(I)$is a fixpoint of T_P.

By lemma 18, each fixpoint of T_P is a model of P. As each model of P contains $P(I)$, each fixpoint of P contains $P(I)$. This proves that $P(I)$is the minimal fixpoint of T_P.

Proof of Theorem 20

Let P be a program and I an input such that $P(I)$exists. We first show by induction on i that $\forall i \geq 0, T_P^i(I)$ is defined and $T_P^i(I) \subseteq P(I)$.

- For $i = 0, T_P^i(I) = I$ is defined and $I \subseteq P(I)$.
- Suppose for $i = n, T_P^n(I)$ is defined and $T_P^n(I) \subseteq P(I)$. Since $T_P(P(I))$ is defined and $T_P^n(I) \subseteq P(I)$, by lemma 16, $T_P^{n+1}(I)$ is also defined and $T_P^{n+1}(I) \subseteq P(I)$.

Then, $\forall i \geq 0, T_P^i(I)$ is defined and is included in $P(I)$. Moreover, since T_P is monotone (lemma 16) and $P(I)$is finite (theorem 15), the sequence $\{T_P^i(I)\}_i$ reaches a fixpoint after a finite number N of steps.

$T_P^N(I)$ is a fixpoint of T_P and is included in $P(I)$. As $P(I)$is the minimal fixpoint of T_P (theorem 19), we have $T_P^N(I) = P(I)$.