THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche LIRMM

Vérification d'une méthodologie pour la conception de systèmes numériques critiques

Présenté par Vincent IAMPIETRO

Le Date de la soutenance

Sous la direction de David Delahaye
et David Andreu

Devant le jury composé de

[Nom Prénom], [Titre], [Labo]	[Statut jury]
[Nom Prénom], [Titre], [Labo]	[Statut jury]
[Nom Prénom], [Titre], [Labo]	[Statut jury]

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

List of Abbreviations

SITPN	**S**ynchronously executed **I**nterpreted **T**ime **P**etri **N**et with priorities
VHDL	**V**ery high speed integrated circuit **H**ardware **D**escription **L**anguage
CIS	**C**omponent **I**nstantiation **S**tatement
PCI	**P**lace **C**omponent **I**nstance
TCI	**T**ransition **C**omponent **I**nstance
GPL	**G**eneric **P**rogramming **L**anguage
HDL	**H**ardware **D**escription **L**anguage
LRM	**L**anguage **R**eference **M**anual
DSL	**D**omain **S**pecific **L**anguage
MDE	**M**odel-**D**riven **E**ngineering

For/Dedicated to/To my...

Chapter 1

Preliminary Notions

In this chapter, we introduce the mathematical formalisms and notations used throughout this thesis to express and formalize our ideas. Also, in the last section, we provide the basics to understand the Coq proof assistant, which is the system we use to write our programs and to mechanize our proofs. This chapter is inspired by the book *The Formal Semantics of Programming Languages: an Introduction*, the courses of the Cambridge of formal semantics, the documentation of the Coq proof assistant, and the Coq programming with dependent-types by Adam Chlipala.

add r

1.1 Mathematical notations

1.1.1 Intuitionistic first order logic

The intuitionistic first-order logic [1] constitutes our framework for the expression and the interpretation of logical formulas. The language to express logical formulas is the same between classical and intuitionistic first-order logic. A logical formula is either:

- $\bot$ (bottom), the always false formula, or $\top$ (top), the always true formula
- a predicate (i.e. an atomic formula). A predicate P possibly takes n parameters as inputs and is interpreted to either true, represented by the $\top$ symbol, or false, represented by the $\bot$ symbol. We write $P(x_0, \ldots, x_n)$ to denote an n-ary predicate P applied to the inputs $x_0, \ldots, x_n$.
- the composition of two subformulas with one of the following binary operators: the conjunction $\wedge$, the disjunction $\vee$, the implication $\Rightarrow$, the double implication $\Leftrightarrow$
- the composition of one subformula with the negation operator $\neg$
- a subformula prefixed by the universal quantifier $\forall$ or the existantial quantifier $\exists$. For instance, the formula $\forall x.P(x)$ denotes the atomic formula $P(x)$ where the parameter x is a universally quantified variable of the formula. As a shorthand notation, we write $\forall x, y, z. \ldots$ to denote $\forall x, \forall y, \forall z. \ldots$. The same stands for the existantial quantifier $\exists$.

The difference between the classical first-order logic and the intuitionistic one relies in the absence of the *law of the excluded middle* in the latter logic. In classical logic, the *law of the*

excluded middle considers that a formula can always be either interpreted as true or false, i.e. a formula is always *decidable* regardless the valuation of its inputs. In intuitionistic logic, such an assumption is discarded. Thus, a formula of the intuitionistic logic can be *undecidable*, i.e. given a valuation of its input, one can not always state that the formula is true or false. This is implication is that one as to exhibit a *explicitly-built* proof to show that a given formula is true. One can not rely on the law of excluded middle to perform a proof by contradiction. As so, the intuitionistic logic is said to be *constructive* in the sense that one as to *construct* a concrete proof object to show that a formula is true. The intuitionistic first-order logic is built in the Coq proof assistant. Thus, a logic formula expressed with the Coq proof assistant is an intuitionistic formula by default. For this reason, we choose the intuitionistic first-order logic as our mathematical framework for the expression of logic formulas.

In what follows, we often give the definition of a predicate by relating its semantics to the semantics of an underlying formula. For instance, we define the predicate $IsEven(n)$, for all $n \in \mathbb{N}$, as follows: $IsEven(n) \equiv \exists k \in \mathbb{N}\ s.t.\ 2k = n$.

1.1.2 Set theory

In this thesis, we use set theory as the base formalism for all our mathematical definitions and proofs. In set theory, a set represents a group of elements called the members of the set. For every set X, we write $x \in X$ to denote that the element x is a member of set X. We note $X \subseteq Y$ the fact that a set X is a subset of set Y such that for all x if $x \in X$ then $x \in Y$. From here, there are multiple ways to define a set.

Extensional definition

A set is defined by *extension* with the enumeration of all its members. For instance, $\{1, 0, -1\}$, $\{a, b, c\}$ or $\{p_0, \ldots, p_n\}$ are all sets defined by extension.

Intensional definition

The intensional definition of a set specifies a given property verified by all the members of the set. For instance, here is the intensional definition of the set of even numbers: $\{n \in \mathbb{N} \mid \exists k \in \mathbb{N}\ s.t.\ n = 2k\}$ or $\{n \in \mathbb{N} \mid IsEven(n)\}$.

Set operators

Set theory also defines some operators to compose sets together. Given two sets A and B, the following sets are formed:

- $A \cup B$ denotes the set formed by the union of the members of A and the members of B, i.e. $A \cup B = \{x \mid x \in A \vee x \in B\}$.
- $A \cap B$ denotes the set formed by the intersection of set A and B, i.e. $A \cap B = \{x \mid x \in A \wedge x \in B\}$.

- $A \setminus B$ denotes the set formed by the elements of set A that are not elements of set B (the difference between set A and B), i.e. $A \setminus B = \{x \mid x \in A \wedge x \notin B\}$.
- $A \times B$ denotes the cartesian product between the elements of set A and set B, i.e. the set of all ordered pairs defined by $\{(x, y) \mid x \in A \wedge y \in B\}$. We generalize the definition to build the set of n-tuples $A_0 \times A_1 \times \cdots \times A_n$ defined by $\{(x_0, (x_1, \ldots (\ldots, x_n) \mid x_0 \in A_0, x_1 \in A_1, \ldots, x_n \in A_n\}$. It is sometimes useful to give a name to the elements of a tuple without refering to their index. In such a case, a tuple is called a record where each element, called a field, has been given an explicit name. This formalism is useful to represent rather complex data structures. For instance, say that we want to represent the set of humans by a triplet composed of the size, weight, and eye color. We can define this set as the set of triplet $\mathbb{R} \times \mathbb{R} \times \{green, blue, brown\}$. If we want to give a concrete name to the elements of the triplet, we can equivalently define such a triplet as a record, written $<size, weight, eye>$, where $size \in \mathbb{R}$, $weight \in \mathbb{R}$ and $eye \in \{green, blue, brown\}$.
- $A \sqcup B$ denotes the set formed by the disjoint union of set A and set B. The disjoint union is obtained by adjoining an index i to the elements of A and an index j to the elements of B such that $i \neq j$. Then, the two sets of couples are joined together to build the disjoint union of A and B. For instance, consider that $A = \{a, b, c\}$ and $B = \{a, b\}$. To obtain the disjoint union $A \sqcup B$, we create the two sets $A_i = \{(i, a), (i, b), (i, c)\}$ and $B_i = \{(j, a), (j, b)\}$, and then join the sets together s.t. $A \sqcup B = \{(i, a), (i, b), (i, c), (j, a), (j, b)\}$. When the two sets A and B are disjoint, i.e. $A \cap B = \emptyset$, then $A \sqcup B$ is isomorphic to $A \cup B$. To stress the fact that we are building a set from the union of two disjoint set, we prefer to use the disjoint union operator. For instance, we write $\mathbb{N} \sqcup \{\infty\}$, instead of $\mathbb{N} \cup \{infty\}$, to denote the set of values ranging from the set of natural numbers with the addition of the infinite value.
- $\mathcal{P}(A)$ denotes the powerset of A defined by all the possible subsets formed with the elements of set A, i.e. $\mathcal{P}(A) = \{X \mid X \subseteq Y\}$.

Relations and functions

A binary relation R between two sets X and Y is a subset of the set of pairs $X \times Y$, i.e. $R \subseteq X \times Y$, or an element of the powerset $\mathcal{P}(X \times Y)$, i.e. $R \in \mathcal{P}(X \times Y)$. We write $R(x, y)$ to denote $(x, y) \in R$. We generalize the definition to n-ary relations. A n-ary relation between sets $X_0, \ldots, X_n$ is a subset of the set of n-tuples $X_0 \times \cdots \times X_n$, i.e. $R \subseteq X_0 \times \cdots \times X_n$, or an element of the powerset $\mathcal{P}(X_0 \times \cdots \times X_n)$, i.e. $R \in \mathcal{P}(X_0 \times \cdots \times X_n)$. We write $R(x_0, \ldots, x_n)$ to denote $(x_0, \ldots, x_n) \in R$.

A partial function f from set X to set Y is a binary relation from X to Y verifying that $\forall x \in X, y, y' \in Y,\ (x, y) \in f \wedge (x, y') \in f \Rightarrow y = y'$, i.e. x appears at most once as the first element of a pair in f. We note $f \in X \nrightarrow Y$ to denote a partial function from X to Y. The set of the first elements of the pairs defined in f is called the *domain* of f, written $\texttt{dom}(f)$. We write it $\texttt{dom}(f) = \{x \mid \exists y \ s.t. (x, y) \in f\}$.

A total function a, or application, from X to Y is a partial function verifying that all the elements of X appear as the first element of a pair in a, i.e. for all $x \in X$, there exists $y \in Y$ such that $(x, y) \in a$. We note $a \in X \to Y$ to denote an application from X to Y.

1.1.3 Rule-based definition of sets

While describing the operational semantics of a subset of the VHDL language in Chapter **??**, we define some of the sets (e.g, simulation relations) with rule instances, also called inference rules or judgments. A rule instance, defining the members of a set A, can take the following form:

$$\frac{}{C} \text{ or } \frac{P_1, \ldots, P_n}{C}$$

The left form of rule instance is called an axiom; it states that $C \in A$. The rule instance $\frac{P_1, \ldots, P_n}{C}$ states $C \in A$ if $P_1, \ldots, P_n$ hold. Also, we say that C is derivable, if $P_1, \ldots, P_n$ are derivable. $P_1, \ldots, P_n$ are the premises of the rule, and C is the conclusion of the rule. The premises $P_1, \ldots, P_n$ hold if there exists a *finite* derivation tree for each one of them. A *finite* derivation tree is obtained by applying the rule instances defining the set until all branches of the derivation reach axioms. For instance, the two following rules define the *IsEven* unary relation that states that a given natural number is even:

$$\frac{}{IsEven(0)} \qquad \frac{IsEven(n-2)}{IsEven(n)}$$

The first rule states as an axiom that 0 is an even number; the second states that for all natural number n, n is an even number if one can derive that fact that $n-2$ is an even number. Thus, we can derive from the previous rules that 4 is an even number by building the following derivation tree:

$$\dfrac{\dfrac{\dfrac{}{IsEven(0)}}{IsEven(2)}}{IsEven(4)}$$

Here, all branches of the tree has reached an axiom, and thus the derivation tree is finite. The further illustrate the use of rule instances in the definition of a set, let us consider the following minimal language of arithmetic expressions expressed in the Backus-Naur form:

$$e \quad ::= \quad n \mid id \mid e_0 + e_1$$

n ranges over the set of natural numbers $\mathbb{N}$; id ranges over the set `string` of non-empty strings (i.e. it is the set of identifiers). To evaluation the arithmetic expressions, we need a state $s \in \texttt{string} \to \mathbb{N}$ that maps each variable identifier to a natural number value. We define the evaluation relation for these arithmetic expressions with the three following rules:

$$\frac{}{s \vdash n \to n}\ \textsc{Nat} \qquad \frac{}{s \vdash id \to s(id)}\ \textsc{Var}\ id \in \texttt{dom}(s) \qquad \frac{s \vdash e_0 \to n \qquad s \vdash e_1 \to m}{s \vdash e_0 + e_1 \to n + m}\ \textsc{Add}$$

Here, the evaluation relation is a subset of the set of triplets $(\texttt{string} \to \mathbb{N}) \times e \times \mathbb{N}$. In the rule instances defining the evaluation relation, the $\vdash$ symbol (pronounced *thesis*) stresses that the left part implies the right part, or it is involved in the evaluation of the right part. For instance, the second rule can be read: in the context of state s, id evaluates to $s(id)$. Thus, when the *context* is not involved in the evaluation of the syntactic constructs at the right side of the $\vdash$ symbol, we permit ourselves to remove the context and the $\vdash$ from the rule instances. For example, we can define to the NAT rule by

$$\frac{}{n \to n}\ \textsc{Nat}$$

as the state s in not involved in the evaluation of expressions that are natural numbers.

Note that in the VAR rule, there appears an extra statement, at the right of the judgment line, called a *side condition*. This is an extra condition that must hold with all the premises of the rule instance, but which is does not generate a derivation tree of its own.

Finally, here is an example of a derivation tree for the evaluation of the expression $x + (y + 1)$ in the context of state $\{(x, 1), (y, 2)\}$:

$$\dfrac{\dfrac{}{\{(x,1),(y,2)\} \vdash x \to 1}\ \textsc{Var}\ x \in \{x,y\} \qquad \dfrac{\dfrac{}{\{(x,1),(y,2)\} \vdash y \to 2}\ \textsc{Var}\ y \in \{x,y\} \qquad \dfrac{}{1 \to 1}\ \textsc{Nat}}{\{(x,1),(y,2)\} \vdash y + 1 \to 3}\ \textsc{Add}}{\{(x,1),(y,2)\} \vdash x + (y + 1) \to 4}\ \textsc{Add}$$

Bibliography

[1] Joan Moschovakis. "Intuitionistic Logic". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Winter 2018. Metaphysics Research Lab, Stanford University, 2018. URL: https://plato.stanford.edu/archives/win2018/entries/logic-intuitionistic/.